PyHelpers

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

Release 1.4.6

Qian Fu

Birmingham Centre for Railway Research and Education

School of Engineering

University of Birmingham

First release: **September 2019**

Last updated: **February 2023**

Table of Contents

List of Figures

Chapter 1

About PyHelpers

PyHelpers is an open-source Python package designed as a lite toolkit for facilitating data (pre)processing. It offers a miscellaneous collection of handy utilities, which could assist us in performing many common data manipulation tasks, such as reading/writing of file-like objects, handling of various types of data (e.g. geographical data and textual data) and communication with relational databases (e.g. PostgreSQL and Microsoft SQL Server).

Chapter 2

Installation

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

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

To install the most recent version of pyhelpers hosted on GitHub:

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

Note:

- If using a virtual environment, make sure it is activated.
- It is recommended to add pip install the option `--upgrade` (or `-U`) to ensure that you are getting the latest stable release of the package.
- Not all dependencies of pyhelpers are enforced to be installed along with the installation of the package. This is intended to optimise the installation requirements. If a ModuleNotFoundError or an ImportError pops out when importing/running a function, first try to install the module(s)/package(s) mentioned in the error message, and then try importing/running the function again.
- For Windows users, pip may possibly fail to install some packages. In such circumstances, try instead to install their .whl files, which can be downloaded from the web page of the unofficial Windows binaries for Python extension packages.
- For more general instructions on the installation of Python packages, please refer to the official guide of Installing Packages.

To check whether pyhelpers has been correctly installed, try to import the package via an interpreter shell:

```
>>> import pyhelpers

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

```
The latest version is: 1.4.6
```

Chapter 3

Modules

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

settings	Settings of working environment.
dirs	Manipulation of directories and/or file paths.
ops	Miscellaneous operations.
store	Saving, loading and other relevant operations of file-like objects.
geom	Manipulation of geometric/geographical data.
text	Manipulation of textual data.
dbms	Communication with databases.

3.1 settings

Settings of working environment.

3.1.1 Configurations

GDAL/OGR

gdal_configurations([reset, ...])	Alter 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*)

Alter some default configuration options of GDAL/OGR drivers.

Parameters

- **reset** (*bool*) – whether to reset to default settings, defaults to `False`
- **max_tmpfile_size** (*int or 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 DB. defaults to `True`

Examples:

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

>>> gdal_configurations()
```

Note: This can be useful when using GDAL to parse a large PBF file. For example, `gdal_configurations()` is applied by default when importing the package pydriosm, which can be used to work with OpenStreetMap data of PBF format.

See also:

- OpenStreetMap XML and PBF
- pydriosm Documentation

3.1.2 Preferences

Matplotlib

mpl_preferences([reset, backend, font_name, ...])	Alter some Matplotlib parameters.

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*)

Alter some Matplotlib parameters.

Parameters

- **backend** (*str or None*) – specify the backend used for rendering and GUI integration, defaults to `None`
- **font_name** (*None or str*) – name of a font to be used, defaults to `'Times New Roman'`
- **font_size** (*int or float*) – font size, defaults to `13`
- **legend_spacing** (*float or int*) – spacing between labels in plot legend, defaults to `0.7`
- **fig_style** (*str or None*) – style of the figure, defaults to `None`
- **reset** (*bool*) – whether to reset to default settings, defaults to `False`

Examples:

```
>>> import numpy as np
>>> 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(figsize=(6, 6))
...     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(loc='best')
...
...     plt.tight_layout()
...
...     plt.show()

>>> example_plot(random_array)
```

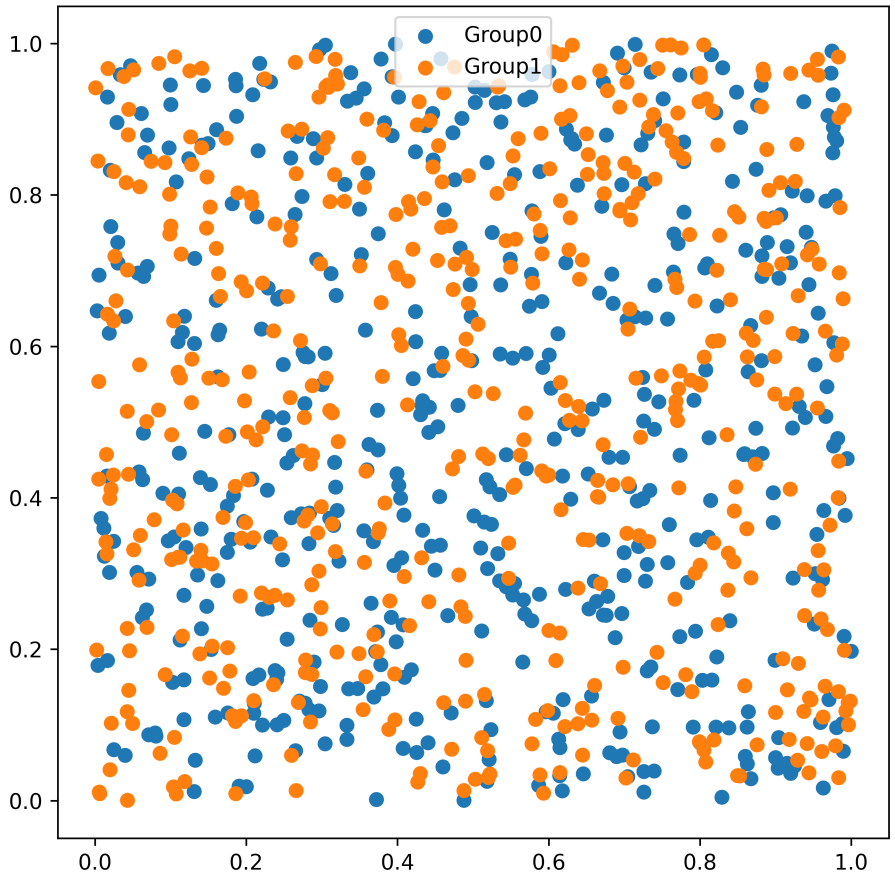

Fig. 1: An example figure, before applying the function *mpl_preferences()*.

```
>>> from pyhelpers.settings import mpl_preferences

>>> mpl_preferences(fig_style='ggplot')

>>> example_plot(random_array)
```

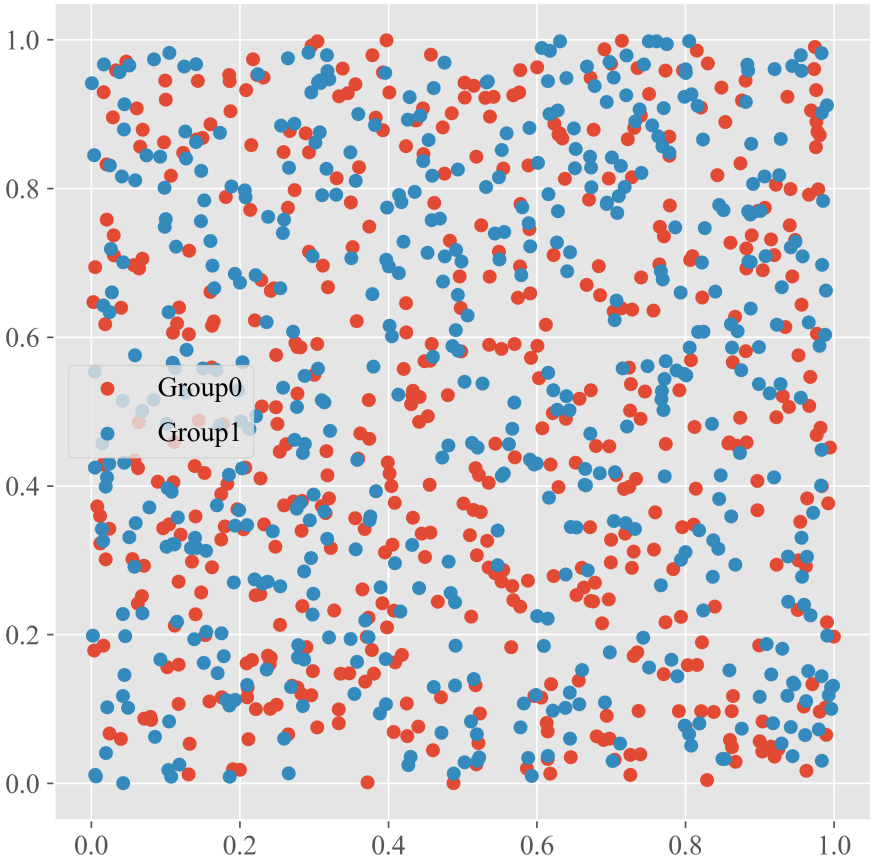

Fig. 2: After applying the function *mpl_preferences()*.

Reset to default settings:

```
>>> mpl_preferences(reset=True)

>>> example_plot(random_array)
```

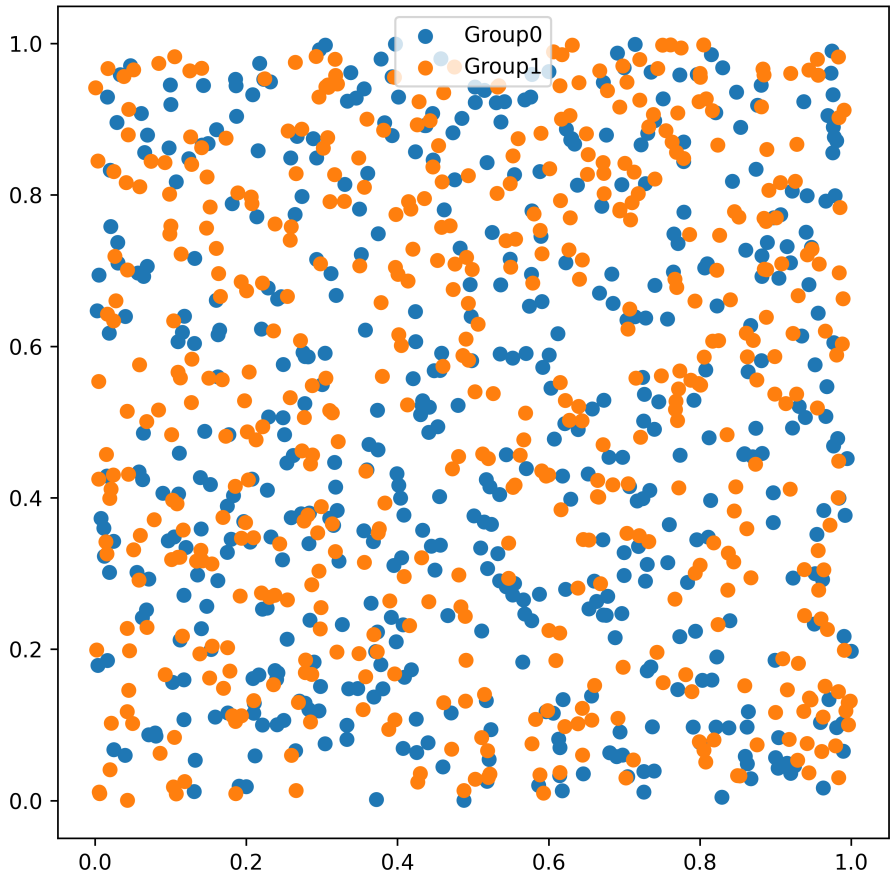

Fig. 3: Resetting the altered parameters to their default values.

NumPy

np_preferences([reset, precision, ...])	Alter some default parameters for displaying NumPy arrays.

np_preferences

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

Alter some default parameters for displaying NumPy arrays.

Parameters

- **reset** (*bool*) – whether to reset to the default print options set by numpy.set_printoptions(), defaults to `False`
- **precision** (*int*) – number of decimal points, which corresponds to `precision` of numpy.set_printoptions(), defaults to `4`

- **line_char** (*int*) – number of characters per line for the purpose of inserting line breaks, which corresponds to `linewidth` of numpy.set_printoptions(), defaults to `120`
- **head_tail** (*int*) – number of array items in summary at beginning (head) and end (tail) of each dimension, which corresponds to `edgeitems` of numpy.set_printoptions(), defaults to `5`
- **formatter** (*dict or None*) – specified format, which corresponds to `formatter` of numpy.set_printoptions(), if `None` (default), fill the empty decimal places with zeros for the specified `precision`

Kwargs

[optional] parameters used by 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,
        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]])
```

Pandas

pd_preferences([reset, max_columns, ...])	Alter parameters of some frequently-used Pandas options and settings for displaying data frame.

pd_preferences

pyhelpers.settings.**pd_preferences**(*reset=False*, *max_columns=100*, *max_rows=20*, *precision=4*, *east_asian_text=False*, *ignore_future_warning=True*)

Alter parameters of some frequently-used Pandas options and settings for displaying data frame.

Parameters

- **reset** (*bool or str*) – whether to reset all to default settings, defaults to `False`
- **max_columns** (*int*) – maximum number of columns, which corresponds to `'display.max_columns'` for pandas.set_option(), defaults to `100`
- **max_rows** (*int*) – maximum number of rows, which corresponds to `'display.max_rows'` for pandas.set_option(), defaults to `20`
- **precision** (*int*) – number of decimal places, which corresponds to `'display.precision'` for pandas.set_option(), defaults to `4`
- **east_asian_text** (*bool*) – whether to adjust the display for east asian texts, defaults to `False`
- **ignore_future_warning** (*bool*) – whether to ignore/suppress future warnings, defaults to `True`

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
          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
```

(continues on next page)

(continued from previous page)

```
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`

3.2 dirs

Manipulation of directories and/or file paths.

3.2.1 Directory navigation

cd(*subdir[, mkdir, cwd, back_check])	Get the full pathname of a directory (or file).
go_from_altered_cwd(dir_name, **kwargs)	Get the full pathname of an altered working directory.
cdd(*subdir[, data_dir, mkdir])	Get the full pathname of a directory (or file) under `data_dir`.
cd_data(*subdir[, data_dir, mkdir])	Get the full pathname of a directory (or file) under `data_dir` of a package.

cd

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

Get the full pathname of a directory (or file).

Parameters

- **subdir** (*str or os.PathLike[str] or bytes or os.Path[bytes]*) – name of a directory or names of directories (and/or a filename)
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **cwd** (*str or os.PathLike[str] or bytes or os.Path[bytes] or None*) – current working directory, defaults to `None`
- **back_check** (*bool*) – whether to check if a parent directory exists, defaults to `False`

- **kwargs** – [optional] parameters (e.g. `mode=0o777`) of os.makedirs

Returns
full pathname of a directory or that of a file

Return type
str

Examples:

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

>>> current_wd = cd()  # Current working directory
>>> os.path.relpath(current_wd)
'.'

>>> # The directory will be created if it does not exist
>>> path_to_tests_dir = cd("tests")
>>> os.path.relpath(path_to_tests_dir)
'tests'

>>> path_to_tests_dir = cd(pathlib.Path("tests"))
>>> os.path.relpath(path_to_tests_dir)
'tests'
```

go_from_altered_cwd

`pyhelpers.dirs.`**`go_from_altered_cwd`**(*dir_name*, ***kwargs*)

Get the full pathname of an altered working directory.

Parameters

- **dir_name** (*str or os.PathLike[str] or bytes or os.Path[bytes]*) – name of a directory
- **kwargs** – [optional] parameters of the function `pyhelpers.dir.cd()`

Returns
full pathname of an altered working directory (changed from the directory `dir_name`)

Return type
str

Examples:

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

>>> init_cwd = os.getcwd()
>>> init_cwd
'<cwd>'
```

(continues on next page)

(continued from previous page)

```
>>> # Change the current working directory to "/new_cwd"
>>> new_cwd = "\new_cwd"
>>> os.mkdir(new_cwd)
>>> os.chdir(new_cwd)

>>> # Get the full path to a folder named "tests"
>>> path_to_tests = go_from_altered_cwd(dir_name="tests")
>>> path_to_tests
'<new_cwd>\tests'

>>> # Get the full path to a directory one level above the current working directory
>>> path_to_tests_ = go_from_altered_cwd(dir_name="\tests")
>>> path_to_tests_ == os.path.join(os.path.dirname(os.getcwd()), "tests")
True

>>> os.chdir(init_cwd)
>>> os.rmdir(new_cwd)
```

cdd

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

Get the full pathname of a directory (or file) under `data_dir`.

Parameters

- **subdir** (*str or os.PathLike[str] or bytes or os.Path[bytes]*) – name of directory or names of directories (and/or a filename)
- **data_dir** (*str or os.PathLike[str] or bytes or os.Path[bytes]*) – name of a directory where data is (or will be) stored, defaults to `"data"`
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **kwargs** – [optional] parameters of the function `pyhelpers.dir.cd()`

Return path

full pathname of a directory (or a file) under `data_dir`

Return type

str

Examples:

```
>>> from pyhelpers.dirs import cdd, delete_dir
>>> import os

>>> path_to_dat_dir = cdd()
>>> # As `mkdir=False`, `path_to_dat_dir` will NOT be created if it doesn't exist
>>> os.path.relpath(path_to_dat_dir)
'data'

>>> path_to_dat_dir = cdd(data_dir="test_cdd", mkdir=True)
>>> # As `mkdir=True`, `path_to_dat_dir` will be created if it doesn't exist
>>> os.path.relpath(path_to_dat_dir)
```

(continues on next page)

(continued from previous page)

```
'test_cdd'

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

>>> # Set `data_dir` to be `"tests"`
>>> path_to_dat_dir = cdd("data", data_dir="test_cdd", mkdir=True)
>>> os.path.relpath(path_to_dat_dir)
'test_cdd\data'

>>> # Delete the "test_cdd" folder and the sub-folder "data"
>>> test_cdd = os.path.dirname(path_to_dat_dir)
>>> delete_dir(test_cdd, verbose=True)
The directory "test_cdd\" is not empty.
Confirmed to delete it
? [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*, ***kwargs*)

Get the full pathname of a directory (or file) under `data_dir` of a package.

Parameters

- **subdir** (*str or os.PathLike[str] or bytes or os.Path[bytes]*) – name of directory or names of directories (and/or a filename)
- **data_dir** (*str or os.PathLike[str] or bytes or os.Path[bytes]*) – name of a directory to store data, defaults to `"data"`
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **kwargs** – [optional] parameters (e.g. `mode=0o777`) of os.makedirs

Returns

full pathname of a directory or that of a file under `data_dir` of a package

Return type

str

Examples:

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

>>> path_to_dat_dir = cd_data("tests", mkdir=False)
```

(continues on next page)

(continued from previous page)

```
>>> os.path.relpath(path_to_dat_dir)
'pyhelpers\data\tests'
```

3.2.2 Directory validation

is_dir(path_to_dir)	Check whether a directory-like string is a (valid) directory name.
validate_dir([path_to_dir, subdir, msg])	Validate the pathname of a directory.

is_dir

pyhelpers.dirs.**is_dir**(*path_to_dir*)

Check whether a directory-like string is a (valid) directory name.

See also [DIRS-IVD-1] and [DIRS-IVD-2].

Parameters
path_to_dir (*str or bytes*) – pathname of a directory

Returns
whether the input is a path-like string that describes a directory name

Return type
bool

Examples:

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

>>> x = "tests"
>>> is_dir(x)
False

>>> x = "/tests"
>>> is_dir(x)
True

>>> x = cd("tests")
>>> is_dir(x)
True
```

validate_dir

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

Validate the pathname of a directory.

Parameters

- **path_to_dir** (*str or os.PathLike[str] or bytes or os.Path[bytes] or None*) – pathname of a data directory, defaults to None
- **subdir** (*str or os.PathLike[str] or bytes or os.Path[bytes]*) – name of a subdirectory to be examined if directory=None, defaults to ""
- **msg** (*str*) – error message if data_dir is not a full pathname, defaults to "Invalid input!"
- **kwargs** – [optional] parameters of the function pyhelpers.dir.cd()

Returns

valid full pathname of a directory

Return type

str

Examples:

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

>>> dat_dir = validate_dir()
>>> os.path.relpath(dat_dir)
'.'

>>> dat_dir = validate_dir("tests")
>>> os.path.relpath(dat_dir)
'tests'

>>> dat_dir = validate_dir(subdir="data")
>>> os.path.relpath(dat_dir)
'data'
```

3.2.3 Directory removal

delete_dir(path_to_dir[, ...])	Delete a directory or directories.

delete_dir

pyhelpers.dirs.**delete_dir**(*path_to_dir*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Delete a directory or directories.

Parameters

- **path_to_dir** (*str or bytes or os.PathLike[str] or os.PathLike[bytes] or collections.abc.Sequence*) – pathname (or pathnames) of a directory (or directories)
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`
- **kwargs** – [optional] parameters of shutil.rmtree or os.rmdir

Examples:

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

>>> 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()

>>> delete_dir(path_to_dir=test_dirs, verbose=True)
To delete the following directories:
    "tests\test_dir0\" (Not empty)
    "tests\test_dir1\" (Not empty)
    "tests\test_dir2\"
? [No]|Yes: yes
Deleting "tests\test_dir0\" ... Done.
Deleting "tests\test_dir1\" ... Done.
Deleting "tests\test_dir2\" ... Done.
```

3.3 ops

Miscellaneous operations.

3.3.1 General use

confirmed([prompt, confirmation_required, resp])	Type to confirm whether to proceed or not.
get_obj_attr(obj[, col_names, as_dataframe])	Get main attributes of an object.
eval_dtype(str_val)	Convert a string to its intrinsic data type.
gps_to_utc(gps_time)	Convert standard GPS time to UTC time.
parse_size(size[, binary, precision])	Parse size from / into readable format of bytes.
get_number_of_chunks(file_or_obj[, ...])	Get total number of chunks of a data file, given a minimum limit of chunk size.
get_relative_path(pathname)	Get the relative or absolute path of `pathname` to the current working directory.
find_executable(app_name[, possibilities])	Get pathname of an executable file for a specified application.
hash_password(password[, salt, salt_size, ...])	Hash a password using hashlib.pbkdf2_hmac.
verify_password(password, salt, key[, ...])	Verify a password given salt and key.

confirmed

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

Type to confirm whether to proceed or not.

See also [OPS-C-1].

Parameters

- **prompt** (*str or None*) – a message that prompts a response (Yes/No), defaults to `None`
- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to `True`
- **resp** (*bool*) – default response, defaults to `False`

Returns

a response

Return type

bool

Examples:

```
>>> from pyhelpers.ops import confirmed

>>> if confirmed(prompt="Testing if the function works?", resp=True):
```

(continues on next page)

(continued from previous page)

```
...     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*)

Get main attributes of an object.

Parameters

- **obj** (*object*) – an object, e.g. a class
- **col_names** (*list*) – a list of column names
- **as_dataframe** (*bool*) – whether to return the data in tabular format, defaults to False

Returns

list or tabular data of the main attributes of the given object

Return type

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',
```

(continues on next page)

(continued from previous page)

```
 'host',
 'port',
 'url',
 'username']
```

eval_dtype

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

Convert a string to its intrinsic data type.

Parameters
str_val (*str*) – a string-type variable

Returns
converted value

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'
```

gps_to_utc

pyhelpers.ops.**gps_to_utc**(*gps_time*)

Convert standard GPS time to UTC time.

Parameters
gps_time (*float*) – standard GPS time

Returns
UTC time

Return type
datetime.datetime

Examples:

```
>>> from pyhelpers.ops import gps_to_utc

>>> utc_dt = gps_to_utc(gps_time=1271398985.7822514)
>>> utc_dt
datetime.datetime(2020, 4, 20, 6, 23, 5, 782251)
```

parse_size

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

Parse size from / into readable format of bytes.

Parameters

- **size** (*str or int or float*) – human- or machine-readable format of size
- **binary** (*bool*) – whether to use binary (i.e. factorized by 1024) representation, defaults to `True`; if `binary=False`, use the decimal (or metric) representation (i.e. factorized by 10 ** 3)
- **precision** (*int*) – number of decimal places (when converting `size` to human-readable format), defaults to `1`

Returns

parsed size

Return type

int or 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, no matter whether `binary` is True or False
>>> 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*)

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

Parameters

- **file_or_obj** (*str*) – absolute path to a file
- **chunk_size_limit** (*int or float or None*) – the minimum limit of file size (in mebibyte i.e. MiB, or megabyte, i.e. MB) above which the function counts how many chunks there could be, defaults to 50;
- **binary** (*bool*) – whether to use binary (i.e. factorized by 1024) representation, defaults to True; if binary=False, use the decimal (or metric) representation (i.e. factorized by 10 ** 3)

Returns

number of chunks

Return type

int or None

Examples:

```
>>> from pyhelpers.ops import get_number_of_chunks
>>> import os

>>> file_path = "C:\Program Files\Python39\python39.pdb"

>>> os.path.getsize(file_path)
13611008
>>> get_number_of_chunks(file_path, chunk_size_limit=2)
7
```

get_relative_path

pyhelpers.ops.**get_relative_path**(*pathname*)

Get the relative or absolute path of pathname to the current working directory.

Parameters

pathname (*str or os.PathLike[str]*) – pathname (of a file or a directory)

Returns

the relative or absolute path of path_to_file to the current working directory

Return type

str or os.PathLike[str]

Examples:

```
>>> from pyhelpers.ops import get_relative_path
>>> import os
```

(continues on next page)

(continued from previous page)

```
>>> rel_pathname = get_relative_path(pathname="")
>>> rel_pathname
''
>>> rel_pathname = get_relative_path(pathname=os.path.join(os.getcwd(), "tests"))
>>> rel_pathname
'tests'

>>> # On Windows OS
>>> rel_pathname = get_relative_path(pathname="C:/Windows")
>>> rel_pathname
"C:/Windows"
```

find_executable

pyhelpers.ops.**find_executable**(*app_name*, *possibilities=None*)

Get pathname of an executable file for a specified application.

Parameters

- **app_name** (*str*) – executable filename of the application that is to be called
- **possibilities** (*list or set or None*) – possible pathnames

Returns

pathname of the specified executable file and whether it exists

Return type

Tuple[str, bool]

Examples:

```
>>> from pyhelpers.ops import find_executable
>>> import os

>>> python_exe = "python.exe"
>>> possible_paths = ["C:\Program Files\Python39", "C:\Python39"]

>>> path_to_python_exe, python_exe_exists = find_executable(python_exe, possible_paths)
>>> os.path.relpath(path_to_python_exe)
'venv\Scripts\python.exe'
>>> python_exe_exists
True

>>> text_exe = "pyhelpers.exe"  # This file does not actually exist
>>> path_to_test_exe, test_exe_exists = find_executable(text_exe, possible_paths)
>>> path_to_test_exe
'pyhelpers.exe'
>>> test_exe_exists
False
```

hash_password

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

Hash a password using hashlib.pbkdf2_hmac.

See also [OPS-HP-1].

Parameters

- **password** (*str or int or float*) – input as a password
- **salt** (*bytes or str*) – random data; when `salt=None` (default), it is generated by os.urandom(), which depends on `salt_size`; see also [OPS-HP-2]
- **salt_size** (*int or None*) – `size` of the function os.urandom(), i.e. the size of a random bytestring for cryptographic use; when `salt_size=None` (default), it uses `128`
- **iterations** (*int or None*) – `size` of the function hashlib.pbkdf2_hmac(), i.e. number of iterations of SHA-256; when `salt_size=None` (default), it uses `100000`
- **ret_hash** (*bool*) – whether to return the salt and key, defaults to `True`
- **kwargs** – [optional] parameters of the function hashlib.pbkdf2_hmac()

Returns

(only when `ret_hash=True`) salt and key

Return type

bytes

Examples:

```
>>> from pyhelpers.ops import hash_password, verify_password

>>> sk = hash_password('test%123', salt_size=16)  # salt and key

>>> salt_data = sk[:16].hex()
>>> key_data = sk[16:].hex()

>>> verify_password('test%123', salt=salt_data, key=key_data)
True
```

verify_password

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

Verify a password given salt and key.

Parameters

- **password** (*str or int or float*) – input as a password
- **salt** (*bytes or str*) – random data; see also [OPS-HP-1]
- **key** (*bytes or str*) – PKCS#5 password-based key (produced by the function hashlib.pbkdf2_hmac())
- **iterations** (*int or None*) – size of the function hashlib.pbkdf2_hmac(), i.e. number of iterations of SHA-256; when `salt_size=None` (default), it uses `100000`

Returns

whether the input password is correct

Return type

bool

See also:

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

3.3.2 Basic data manipulation

Iterable

loop_in_pairs(iterable)	Get every pair (current, next).
split_list_by_size(lst, sub_len)	Split a list into (evenly sized) sub-lists.
split_list(lst, num_of_sub)	Split a list into a number of equally-sized sub-lists.
split_iterable(iterable, chunk_size)	Split a list into (evenly sized) chunks.
update_dict(dictionary, updates[, inplace])	Update a (nested) dictionary or similar mapping.
update_dict_keys(dictionary[, replacements])	Update keys in a (nested) dictionary.
get_dict_values(key, dictionary)	Get all values in a (nested) dictionary for a given key.
remove_dict_keys(dictionary, *keys)	Remove multiple keys from a dictionary.
compare_dicts(dict1, dict2)	Compare the difference between two dictionaries.
merge_dicts(*dicts)	Merge multiple dictionaries.

loop_in_pairs

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

Get every pair (current, next).

Parameters
iterable (*Iterable*) – iterable object

Returns
a zip-type variable

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*)

Split a list into (evenly sized) sub-lists.

See also [OPS-SLBS-1].

Parameters

- **lst** (*list*) – a list of any
- **sub_len** (*int*) – length of a sub-list

Returns
a sequence of sub_len-sized sub-lists 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*)

Split a list into a number of equally-sized sub-lists.

See also [OPS-SL-1].

Parameters

- **lst** (*list*) – a list of any
- **num_of_sub** (*int*) – number of sub-lists

Returns

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*)

Split a list into (evenly sized) chunks.

See also [OPS-SI-1].

Parameters

- **iterable** (*Iterable*) – iterable object
- **chunk_size** (*int*) – length of a chunk

Returns

a sequence of equally-sized chunks from iterable

Return type

Generator[*Iterable*]

Examples:

```
>>> from pyhelpers.ops import split_iterable
>>> import pandas

>>> iterable_1 = list(range(0, 10))
```

(continues on next page)

(continued from previous page)

```
>>> 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)

>>> 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*)

Update a (nested) dictionary or similar mapping.

See also [OPS-UD-1].

Parameters

- **dictionary** (*dict*) – a (nested) dictionary that needs to be updated
- **updates** (*dict*) – a dictionary with new data
- **inplace** (*bool*) – whether to replace the original `dictionary` with the updated one, defaults to `False`

Returns

an 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'}}

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

(continues on next page)

(continued from previous page)

```
>>> upd_dict
{'key': {'k1': 'v1', 'k2': 'v2'}}
```

update_dict_keys

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

Update keys in a (nested) dictionary.

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

Parameters

- **dictionary** (*dict*) – a (nested) dictionary in which certain keys are to be updated
- **replacements** (*dict or None*) – a dictionary in the form of {<current_key>: <new_key>}, describing which keys are to be updated, defaults to None

Returns

an updated dictionary

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*)

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

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

Parameters

- **key** (*any*) – any that can be the key of a dictionary
- **dictionary** (*dict*) – a (nested) dictionary

Returns

all values of the `key` within the given `target_dict`

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*)

Remove multiple keys from a dictionary.

Parameters

- **dictionary** (*dict*) – a dictionary
- **keys** (*any*) – (a sequence of) any that can be the key of a 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*)

Compare the difference between two dictionaries.

See also [OPS-CD-1].

Parameters

- **dict1** (*dict*) – a dictionary
- **dict2** (*dict*) – another dictionary

Returns

in comparison to `dict1`, the main difference on `dict2`, including: modified items, keys that are the same, keys where values remain unchanged, new keys and keys that are removed

Return type

Tuple[dict, list]

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']
```

(continues on next page)

(continued from previous page)

```
>>> k_unchanged
['b']
>>> k_new
['d']
>>> k_removed
['a']
```

merge_dicts

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

Merge multiple dictionaries.

Parameters
dicts (*dict*) – (one or) multiple dictionaries

Returns
a single dictionary containing all elements of the input

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

Tabular data

detect_nan_for_str_column(data_frame[, ...])	Detect if a str type column contains NaN when reading csv files.
create_rotation_matrix(theta)	Create a rotation matrix (counterclockwise).
dict_to_dataframe(input_dict[, k, v])	Convert a dictionary to a data frame.
parse_csr_matrix(path_to_csr[, verbose])	Load in a compressed sparse row (CSR) or compressed row storage (CRS).
swap_cols(array, c1, c2[, as_list])	Swap positions of two columns in an array.
swap_rows(array, r1, r2[, as_list])	Swap positions of two rows in an array.
np_shift(array, step[, fill_value])	Shift an array by desired number of rows.

detect_nan_for_str_column

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

Detect if a str type column contains NaN when reading csv files.

Parameters

- **data_frame** (*pandas.DataFrame*) – a data frame to be examined
- **column_names** (*None or collections.abc.Iterable*) – a sequence of column names, if None (default), all columns

Returns

position index of the column that contains NaN

Return type

Generator[*Iterable*]

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
>>> dat
            Easting  Northing
City
London       530034  180381.0
Birmingham   406689  286822.0
Manchester   383819  398052.0
Leeds        582044       NaN
```

(continues on next page)

(continued from previous page)

```
>>> 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*)

Create a rotation matrix (counterclockwise).

Parameters
theta (*int or float*) – rotation angle (in radian)

Returns
a rotation matrix of shape (2, 2)

Return type
numpy.ndarray

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'*)

Convert a dictionary to a data frame.

Parameters

- **input_dict** (*dict*) – a dictionary to be converted to a data frame
- **k** (*str*) – column name for keys
- **v** (*str*) – column name for values

Returns
a data frame converted from the input_dict

Return type
pandas.DataFrame

Examples:

```
>>> from pyhelpers.ops import dict_to_dataframe

>>> test_dict = {'a': 1, 'b': 2}
```

(continues on next page)

(continued from previous page)

```
>>> dat = dict_to_dataframe(input_dict=test_dict)
>>> dat
  key  value
0   a      1
1   b      2
```

parse_csr_matrix

pyhelpers.ops.**parse_csr_matrix**(*path_to_csr*, *verbose=False*, ***kwargs*)

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

Parameters

- **path_to_csr** (*str*) – path where a CSR file (e.g. with a file extension ".npz") is saved
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`
- **kwargs** – [optional] parameters of numpy.load

Returns

a compressed sparse row

Return type

scipy.sparse.csr.csr_matrix

Examples:

```
>>> from pyhelpers.ops import parse_csr_matrix
>>> from pyhelpers.dirs import cd
>>> from scipy.sparse import csr_matrix, save_npz

>>> data_ = [1, 2, 3, 4, 5, 6]
>>> indices_ = [0, 2, 2, 0, 1, 2]
>>> indptr_ = [0, 2, 3, 6]

>>> csr_m = csr_matrix((data_, indices_, indptr_), shape=(3, 3))
>>> csr_m
<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")
>>> save_npz(path_to_csr_npz, csr_m)

>>> parsed_csr_mat = parse_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)
>>> (parsed_csr_mat != csr_m).count_nonzero() == 0
True

>>> (parsed_csr_mat != csr_m).nnz == 0
True
```

swap_cols

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

Swap positions of two columns in an array.

Parameters

- **array** (*numpy.ndarray*) – an array
- **c1** (*int*) – index of a column
- **c2** (*int*) – index of another column
- **as_list** (*bool*) – whether to return a list

Returns

a new array/list in which the positions of the c1-th and c2-th columns are swapped

Return type

numpy.ndarray or 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*)

Swap positions of two rows in an array.

Parameters

- **array** (*numpy.ndarray*) – an array
- **r1** (*int*) – index of a row

- **r2** (*int*) – index of another row
- **as_list** (*bool*) – whether to return a list

Returns
a new array/list in which the positions of the r1-th and r2-th rows are swapped

Return type
numpy.ndarray or 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*)

Shift an array by desired number of rows.

See also [OPS-NS-1]

Parameters

- **array** (*numpy.ndarray*) – an array of numbers
- **step** (*int*) – number of rows to shift
- **fill_value** (*float or int*) – values 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]])
```

3.3.3 Basic computation

get_extreme_outlier_bounds(num_dat[, k])	Get upper and lower bounds for extreme outliers.
interquartile_range(num_dat)	Calculate interquartile range.
find_closest_date(date, lookup_dates[, ...])	Find the closest date of a given one from a list of dates.

get_extreme_outlier_bounds

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

Get upper and lower bounds for extreme outliers.

Parameters

- **num_dat** (*array-like*) – an array of numbers
- **k** (*float*, *int*) – a scale coefficient associated with interquartile range, defaults to `1.5`

Returns

lower and upper bound

Return type

tuple

Examples:

```
>>> from pyhelpers.ops import get_extreme_outlier_bounds
>>> import pandas

>>> data = pandas.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
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)
```

interquartile_range

pyhelpers.ops.**interquartile_range**(*num_dat*)

Calculate interquartile range.

This function may be an alternative to scipy.stats.iqr.

Parameters
num_dat (*numpy.ndarry or list or tuple*) – an array of numbers

Returns
interquartile range of num_dat

Return type
float

Examples:

```
>>> from pyhelpers.ops import interquartile_range
```

(continues on next page)

(continued from previous page)

```
>>> data = list(range(100))

>>> iqr_result = interquartile_range(data)
>>> iqr_result
49.5
```

find_closest_date

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

Find the closest date of a given one from a list of dates.

Parameters

- **date** (*str or datetime.datetime*) – a date
- **lookup_dates** (*Iterable*) – an array of dates
- **as_datetime** (*bool*) – whether to return a datetime.datetime-formatted date, defaults to `False`
- **fmt** (*str*) – datetime format, defaults to `'%Y-%m-%d %H:%M:%S.%f'`

Returns

the date that is closest to the given `date`

Return type

str or datetime.datetime

Examples:

```
>>> from pyhelpers.ops import find_closest_date
>>> import pandas

>>> example_dates = pandas.date_range('2019-01-02', '2019-12-31')
>>> example_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')

>>> example_date = '2019-01-01'
>>> closest_example_date = find_closest_date(example_date, example_dates)
>>> closest_example_date
'2019-01-02 00:00:00.000000'

>>> example_date = pandas.to_datetime('2019-01-01')
>>> closest_example_date = find_closest_date(example_date, example_dates, as_datetime=True)
>>> closest_example_date
Timestamp('2019-01-02 00:00:00', freq='D')
```

3.3.4 Graph plotting

cmap_discretisation(cmap, n_colours)	Create a discrete colour ramp.
colour_bar_index(cmap, n_colours[, labels])	Create a colour bar.

cmap_discretisation

pyhelpers.ops.**cmap_discretisation**(*cmap*, *n_colours*)

Create a discrete colour ramp.

See also [OPS-CD-1].

Parameters

- **cmap** (*matplotlib.colors.ListedColormap or matplotlib.colors.LinearSegmentedColormap*) – a colormap instance, e.g. built-in colormaps that is accessible via matplotlib.cm.get_cmap
- **n_colours** (*int*) – number of colours

Returns

a discrete colormap from (the continuous) cmap

Return type

matplotlib.colors.LinearSegmentedColormap

Examples:

```
>>> from pyhelpers.ops import cmap_discretisation
>>> import matplotlib.cm
>>> import matplotlib.pyplot as plt
>>> import numpy

>>> cm_accent = cmap_discretisation(matplotlib.cm.get_cmap('Accent'), n_colours=5)
>>> cm_accent.name
'Accent_5'

>>> fig, ax = plt.subplots(figsize=(10, 2))

>>> ax.imshow(numpy.resize(range(100), (5, 100)), cmap=cm_accent, interpolation='nearest')

>>> plt.axis('off')
>>> plt.tight_layout()
>>> plt.show()
```

The exmaple is illustrated in Fig. 4:

Fig. 4: An example of discrete colour ramp, created by the function *cmap_discretisation()*.

```
>>> plt.close()
```

colour_bar_index

pyhelpers.ops.**colour_bar_index**(*cmap*, *n_colours*, *labels=None*, ***kwargs*)

Create a colour bar.

To stop making off-by-one errors. Takes a standard colour ramp, and discretizes it, then draws a colour bar with correctly aligned labels.

See also [OPS-CBI-1].

Parameters

- **cmap** (*matplotlib.colors.ListedColormap*) – a colormap instance, e.g. built-in colormaps that is accessible via matplotlib.cm.get_cmap
- **n_colours** (*int*) – number of colours
- **labels** (*list or None*) – a list of labels for the colour bar, defaults to None
- **kwargs** – [optional] parameters of matplotlib.pyplot.colorbar

Returns

a colour bar object

Return type

matplotlib.colorbar.Colorbar

Examples:

```
>>> from pyhelpers.ops import colour_bar_index
>>> import matplotlib
>>> import matplotlib.pyplot as plt
>>> import matplotlib.cm

>>> matplotlib.use('TkAgg')

>>> plt.figure(figsize=(2, 6))

>>> cbar = colour_bar_index(cmap=matplotlib.cm.get_cmap('Accent'), n_colours=5)

>>> plt.xticks(fontsize=12)
>>> plt.yticks(fontsize=12)
>>> cbar.ax.tick_params(labelsize=14)

>>> # plt.axis('off')
>>> plt.tight_layout()
>>> plt.show()
```

The above example is illustrated in Fig. 5:

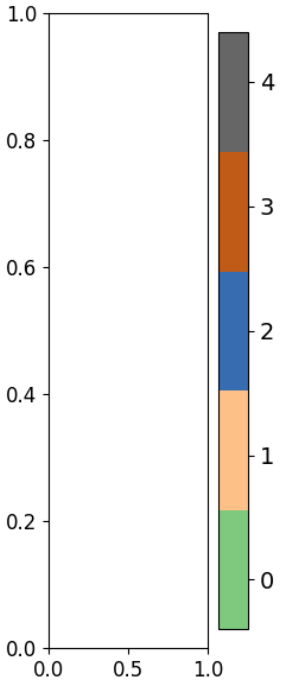

Fig. 5: An example of colour bar with numerical index, created by the function *colour_bar_index()*.

```
>>> plt.figure(figsize=(2, 6))

>>> labels_ = list('abcde')
>>> cbar = colour_bar_index(matplotlib.cm.get_cmap('Accent'), n_colours=5, labels=labels_)

>>> plt.xticks(fontsize=12)
>>> plt.yticks(fontsize=12)
>>> cbar.ax.tick_params(labelsize=14)

>>> # plt.axis('off')
>>> plt.tight_layout()
>>> plt.show()
```

This second example is illustrated in Fig. 6:

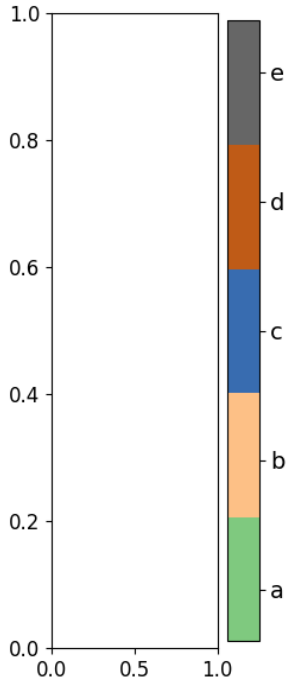

Fig. 6: An example of colour bar with textual index, created by the function *colour_bar_index()*.

```
>>> plt.close(fig='all')
```

3.3.5 Web data extraction

is_network_connected()	Check whether the current machine can connect to the Internet.
is_url(url[, partially])	Check whether `url` is a valid URL.
is_url_connectable(url)	Check whether the current machine can connect to a given URL.
is_downloadable(url[, request_field])	Check whether a URL leads to a web page where there is downloadable contents.
init_requests_session(url[, max_retries, ...])	Instantiate a requests session.
load_user_agent_strings([shuffled, ...])	Load user-agent strings of popular browsers.
get_user_agent_string([fancy])	Get a random user-agent string of a certain browser.
fake_requests_headers([randomized])	Make a fake HTTP headers for requests.get.
download_file_from_url(url, path_to_file[, ...])	Download an object available at a valid URL.

is_network_connected

pyhelpers.ops.**is_network_connected**()

Check whether the current machine can connect to the Internet.

Returns
whether the Internet connection is currently working

Return type
bool

Examples:

```
>>> from pyhelpers.ops import is_network_connected

>>> is_network_connected()  # assuming the machine is currently connected to the Internet
True
```

is_url

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

Check whether `url` is a valid URL.

See also [OPS-IU-1]

Parameters

- **url** (*str*) – a string-type variable
- **partially** (*bool*) – whether to consider the input as partially valid, defaults to `False`

Returns
whether `url` is a valid URL

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)
True

>>> is_url(url='github.com')
False

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

(continues on next page)

(continued from previous page)

```
True

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

is_url_connectable

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

Check whether the current machine can connect to a given URL.

Parameters
url (*str*) – a (valid) URL

Returns
whether the machine can currently connect to the given URL

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
```

is_downloadable

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

Check whether a URL leads to a web page where there is downloadable contents.

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] parameters of requests.head

Returns
whether the `url` leads to downloadable contents

Return type
bool

Examples:

```
>>> from pyhelpers.ops import is_downloadable

>>> logo_url = 'https://www.python.org/static/community_logos/python-logo-master-v3-TM.png'
>>> is_downloadable(logo_url)
True

>>> google_url = 'https://www.google.co.uk/'
>>> is_downloadable(google_url)
False
```

init_requests_session

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

Instantiate a requests session.

Parameters

- **url** (*str*) – a valid URL
- **max_retries** (*int*) – maximum number of retries, defaults to `5`
- **backoff_factor** (*float*) – `backoff_factor` of urllib3.util.retry.Retry, defaults to `0.1`
- **retry_status** – a list of HTTP status codes that force to retry downloading, inherited from `status_forcelist` of urllib3.util.retry.Retry; when `retry_status='default'`, the list defaults to `[429, 500, 502, 503, 504]`
- **kwargs** – [optional] parameters of urllib3.util.retry.Retry

Returns

a requests session

Return type

requests.Session

Examples:

```
>>> from pyhelpers.ops import init_requests_session

>>> logo_url = 'https://www.python.org/static/community_logos/python-logo-master-v3-TM.png'

>>> s = init_requests_session(logo_url)

>>> type(s)
requests.sessions.Session
```

load_user_agent_strings

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

Load user-agent strings of popular browsers.

The current version collects 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 make a list of all available user-agent strings, defaults to False
- **update** (*bool*) – whether to update the backup data of user-agent strings, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

a dictionary of user agent strings for popular browsers

Return type

dict or list

Examples:

```
>>> from pyhelpers.ops import load_user_agent_strings

>>> uas = load_user_agent_strings()

>>> list(uas.keys())
['Chrome', 'Firefox', 'Safari', 'Edge', 'Internet Explorer', 'Opera']
>>> type(uas['Chrome'])
list
>>> uas['Chrome'][0]
'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/...

>>> uas_list = load_user_agent_strings(shuffled=True, flattened=True)
>>> type(uas_list)
list
>>> uas_list[0]  # a random one
'Mozilla/5.0 (Windows NT 6.0) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.792.0 Saf...
```

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*)

Get a random user-agent string of a certain browser.

Parameters

- **fancy** – name of a preferred browser, defaults to `None`; options include `'Chrome'`, `'Firefox'`, `'Safari'`, `'Edge'`, `'Internet Explorer'` and `'Opera'`; if `fancy=None`, the function returns a user-agent string of a randomly-selected browser among all the available options
- **kwargs** – [optional] parameters of the function `pyhelpers.ops.get_user_agent_strings()`

Type
fancy: None or str

Returns
a user-agent string of a certain 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
'Opera/7.01 (Windows 98; U)  [en]'

>>> # Get a random Chrome user-agent string
>>> uas_1 = get_user_agent_string(fancy='Chrome')
>>> uas_1
'Mozilla/5.0 (Windows NT 6.0; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.9...
```

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*)

Make a fake HTTP headers for requests.get.

Parameters

- **randomized** (*bool*) – whether to use a user-agent string randomly selected from among all available data of several popular browsers, defaults to `True`; if `randomized=False`, the function uses a random Chrome user-agent string

- **kwargs** – [optional] parameters of the function *pyhelpers.ops.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/525.19 (KHTML...

>>> 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 (KHTML,...
```

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.

download_file_from_url

pyhelpers.ops.**download_file_from_url**(*url*, *path_to_file*, *if_exists='replace'*, *max_retries=5*, *random_header=True*, *verbose=False*, *requests_session_args=None*, *fake_headers_args=None*, ***kwargs*)

Download an object available at a valid URL.

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

Parameters

- **url** (*str*) – valid URL to a web resource
- **path_to_file** (*str or os.PathLike[str]*) – a path where the downloaded object is saved as, or a filename
- **if_exists** (*str*) – given that the specified file already exists, options include `'replace'` (default, continuing to download the requested file and replace the existing one at the specified path) and `'pass'` (cancelling the download)
- **max_retries** (*int*) – maximum number of retries, defaults to `5`

- **random_header** (*bool*) – whether to go for a random agent, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **requests_session_args** (*dict or None*) – [optional] parameters of the function *pyhelpers.ops.init_requests_session()*, defaults to `None`
- **fake_headers_args** (*dict or None*) – [optional] parameters of the function *pyhelpers.ops.fake_requests_headers()*, defaults to `None`
- **kwargs** – [optional] parameters of requests.Session.get()

Examples:

```
>>> from pyhelpers.ops import download_file_from_url
>>> from pyhelpers.dirs import cd
>>> from PIL import Image
>>> import os

>>> logo_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(logo_url, path_to_img)

>>> # If download is successful, check again:
>>> os.path.exists(path_to_img)
True

>>> img = Image.open(path_to_img)
>>> img.show()  # as illustrated below
```

Fig. 7: The Python Logo.

Note:

- When `verbose=True`, the function requires tqdm.

3.4 store

Saving, loading and other relevant operations of file-like objects.

3.4.1 Data saving

save_pickle(pickle_data, path_to_pickle[, ...])	Save data to a Pickle file.
save_spreadsheet(spreadsheet_data, ...[, ...])	Save data to a CSV or an Microsoft Excel file.
save_spreadsheets(spreadsheets_data, ...[, ...])	Save data to a multi-sheet Microsoft Excel file.
save_json(json_data, path_to_json[, engine, ...])	Save data to a JSON file.
save_joblib(joblib_data, path_to_joblib[, ...])	Save data to a Joblib file.
save_feather(feather_data, path_to_feather)	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).
save_fig(path_to_fig_file[, dpi, verbose, ...])	Save a figure object to a file of a supported file format.
save_web_page_as_pdf(web_page, path_to_pdf)	Save a web page as a PDF file by wkhtmltopdf.
save_data(data, path_to_file[, err_warning, ...])	Save data to a file of a specific format.

save_pickle

pyhelpers.store.**save_pickle**(*pickle_data*, *path_to_pickle*, *verbose=False*, ***kwargs*)

Save data to a Pickle file.

Parameters

- **pickle_data** (*any*) – data that could be dumped by the built-in module pickle.dump
- **path_to_pickle** (*str or os.PathLike[str]*) – path where a pickle file is saved
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of pickle.dump

Examples:

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

>>> pickle_dat = 1

>>> pickle_pathname = cd("tests\data", "dat.pickle")
>>> save_pickle(pickle_dat, pickle_pathname, verbose=True)
```

(continues on next page)

(continued from previous page)

```
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" at "tests\data\" ... Done.
```

See also:

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

save_spreadsheet

pyhelpers.store.**save_spreadsheet**(*spreadsheet_data*, *path_to_spreadsheet*, *index=False*, *engine=None*, *delimiter=','*, *verbose=False*, ***kwargs*)

Save data to a CSV or an Microsoft Excel file.

The file extension can be *".txt"*, *".csv"*, *".xlsx"* or *".xls"*; engines can include: xlsxwriter (for .xlsx) and openpyxl (for .xlsx/.xlsm).

Parameters

- **spreadsheet_data** (*pandas.DataFrame*) – data that could be saved as a spreadsheet (e.g. with a file extension ".xlsx" or ".csv")
- **path_to_spreadsheet** (*str or os.PathLike[str] or None*) – path where a spreadsheet is saved
- **index** (*bool*) – whether to include the index as a column, defaults to `False`
- **engine** (*str or None*) – options include `'openpyxl'` for latest Excel file formats, `'xlrd'` for .xls `'odf'` for OpenDocument file formats (.odf, .ods, .odt), and `'pyxlsb'` for Binary Excel files; defaults to `None`
- **delimiter** (*str*) – separator for saving a *".xlsx"* (or *".xls"*) file as a *".csv"* file, defaults to `','`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of 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.
```

save_spreadsheets

pyhelpers.store.**save_spreadsheets**(*spreadsheets_data*, *path_to_spreadsheet*, *sheet_names*, *mode='w'*, *if_sheet_exists=None*, *verbose=False*, ***kwargs*)

Save data to a multi-sheet Microsoft Excel file.

The file extension can be *".xlsx"* or *".xls"*.

Parameters

- **spreadsheets_data** (*list or tuple or iterable*) – a sequence of pandas.DataFrame
- **path_to_spreadsheet** (*str or os.PathLike[str]*) – path where a spreadsheet is saved
- **sheet_names** (*list or tuple or iterable*) – all sheet names of an Excel workbook
- **mode** (*str*) – mode to write to an Excel file; `'w'` (default) for 'write' and `'a'` for 'append'
- **if_sheet_exists** (*None or str*) – indicate the behaviour when trying to write to an existing sheet; see also the parameter `if_sheet_exists` of pandas.ExcelWriter
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of pandas.DataFrame.to_excel

Examples:

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

>>> dat1 = example_dataframe()  # Get an example dataframe
>>> dat1
            Longitude   Latitude
City
London      -0.127647  51.507322
Birmingham  -1.902691  52.479699
Manchester  -2.245115  53.479489
Leeds       -1.543794  53.797418

>>> dat2 = dat1.T
>>> dat2
City          London  Birmingham  Manchester      Leeds
Longitude  -0.127647   -1.902691   -2.245115  -1.543794
Latitude   51.507322   52.479699   53.479489  53.797418

>>> dat = [dat1, dat2]
>>> sheets = ['TestSheet1', 'TestSheet2']
>>> pathname = cd("tests\data", "dat.xlsx")

>>> save_spreadsheets(dat, pathname, sheets, verbose=True)
Saving "dat.xlsx" to "tests\data\" ...
    'TestSheet1' ... Done.
    'TestSheet2' ... Done.

>>> save_spreadsheets(dat, pathname, sheets, 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_spreadsheets(dat, pathname, sheets, 'a', if_sheet_exists='replace', verbose=True)
Updating "dat.xlsx" at "tests\data\" ...
    'TestSheet1' ... Done.
    'TestSheet2' ... Done.

>>> save_spreadsheets(dat, pathname, sheets, '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**(*json_data*, *path_to_json*, *engine=None*, *verbose=False*, ***kwargs*)

Save data to a JSON file.

Parameters

- **json_data** (*any json data*) – data that could be dumped by as a JSON file
- **path_to_json** (*str or os.PathLike[str]*) – path where a json file is saved

- **engine** (*str or None*) – an open-source module used for JSON serialization, options include None (default, for the built-in json module), 'ujson' (for UltraJSON), 'orjson' (for orjson) and 'rapidjson' (for python-rapidjson)
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **kwargs** – [optional] parameters of json.dump() (if engine=None), orjson.dumps() (if engine='orjson'), ujson.dump() (if engine='ujson') or 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" at "tests\data\" ... Done.

>>> save_json(json_dat, json_pathname, engine='orjson', verbose=True)
Updating "dat.json" at "tests\data\" ... Done.

>>> save_json(json_dat, json_pathname, engine='ujson', indent=4, verbose=True)
Updating "dat.json" at "tests\data\" ... Done.

>>> save_json(json_dat, json_pathname, engine='rapidjson', indent=4, verbose=True)
Updating "dat.json" at "tests\data\" ... Done.
```

See also:

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

save_joblib

pyhelpers.store.**save_joblib**(*joblib_data*, *path_to_joblib*, *verbose=False*, ***kwargs*)

Save data to a Joblib file.

Parameters

- **joblib_data** (*any*) – data that could be dumped by joblib.dump
- **path_to_joblib** (*str or os.PathLike[str]*) – path where a pickle file is saved
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **kwargs** – [optional] parameters of joblib.dump

Examples:

```
>>> from pyhelpers.store import save_joblib
>>> from pyhelpers.dirs import cd
>>> from pyhelpers._cache import example_dataframe
>>> import numpy as np

>>> 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:
>>> np.random.seed(0)
>>> joblib_dat = np.random.rand(100, 100)
>>> 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],
       ...,
       [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]])
```

(continues on next page)

(continued from previous page)

```
>>> save_joblib(joblib_dat, joblib_pathname, verbose=True)
Updating "dat.joblib" at "tests\data\" ... Done.
```

See also:

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

save_feather

pyhelpers.store.**save_feather**(*feather_data*, *path_to_feather*, *index=False*, *verbose=False*, ***kwargs*)

Save a dataframe to a Feather file.

Parameters

- **feather_data** (*pandas.DataFrame*) – a dataframe to be saved as a feather-formatted file
- **path_to_feather** (*str or os.PathLike[str]*) – path where a feather file is saved
- **index** (*bool*) – whether to include the index as a column, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of 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.

>>> save_feather(feather_dat, feather_pathname, index=True, verbose=True)
Updating "dat.feather" at "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*, *verbose=False*, *inkscape_exe=None*, ***kwargs*)

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

Parameters

- **path_to_svg** (*str*) – path where a .svg file is saved
- **path_to_emf** (*str*) – path where a .emf file is saved
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **inkscape_exe** (*str or None*) – absolute path to 'inkscape.exe', defaults to None; when inkscape_exe=None, use the default installation path, e.g. (on Windows) "*C:\Program Files\Inkscape\bin\inkscape.exe*" or "*C:\Program Files\Inkscape\inkscape.exe*"
- **kwargs** – [optional] parameters of subprocess.run

Examples:

```
>>> from pyhelpers.store import save_svg_as_emf
>>> from pyhelpers.dirs import cd
>>> from pyhelpers.settings import mpl_preferences
>>> import matplotlib.pyplot as plt

>>> mpl_preferences()

>>> x, y = (1, 1), (2, 2)

>>> plt.figure()
>>> plt.plot([x[0], y[0]], [x[1], y[1]])
>>> plt.show()
```

The above exmaple is illustrated in Fig. 8:

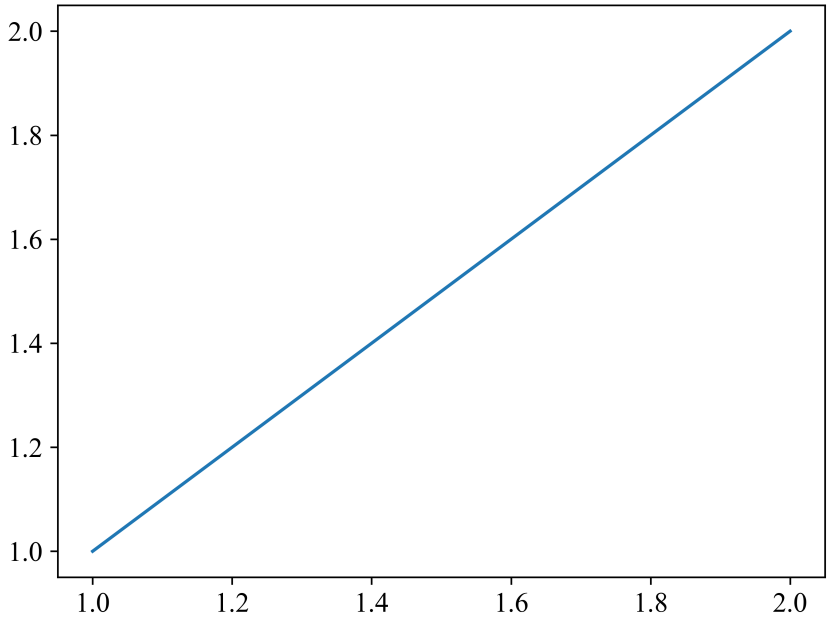

Fig. 8: An example figure created for the function *save_svg_as_emf()*.

```
>>> img_dir = cd("tests\images")

>>> svg_file_pathname = cd(img_dir, "store-save_fig-demo.svg")
>>> plt.savefig(svg_file_pathname)  # Save the figure as a .svg file

>>> emf_file_pathname = cd(img_dir, "store-save_fig-demo.emf")
>>> save_svg_as_emf(svg_file_pathname, emf_file_pathname, verbose=True)
Saving the .svg file as "tests\images\store-save_fig-demo.emf" ... Done.

>>> plt.close()
```

save_fig

pyhelpers.store.**save_fig**(*path_to_fig_file*, *dpi=None*, *verbose=False*, *conv_svg_to_emf=False*, ***kwargs*)

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

This function relies on matplotlib.pyplot.savefig (and Inkscape).

Parameters

- **path_to_fig_file** (*str or os.PathLike[str]*) – path where a figure file is saved
- **dpi** (*int or None*) – the resolution in dots per inch; if None (default), use rcParams['savefig.dpi']

- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **conv_svg_to_emf** (*bool*) – whether to convert a .svg file to a .emf file, defaults to `False`
- **kwargs** – [optional] parameters of matplotlib.pyplot.savefig

Examples:

```
>>> from pyhelpers.store import save_fig
>>> from pyhelpers.dirs import cd
>>> from pyhelpers.settings import mpl_preferences
>>> import matplotlib.pyplot as plt

>>> mpl_preferences()

>>> x, y = (1, 1), (2, 2)

>>> plt.figure()
>>> plt.plot([x[0], y[0]], [x[1], y[1]])
>>> plt.show()
```

The above exmaple is illustrated in Fig. 9:

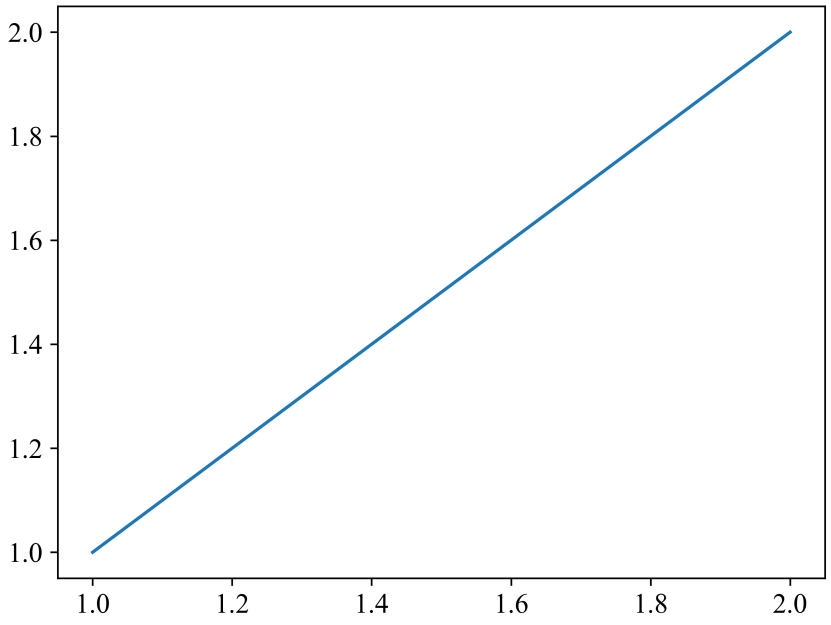

Fig. 9: An example figure created for the function *save_fig()*.

```
>>> img_dir = cd("tests\images")

>>> png_file_pathname = cd(img_dir, "store-save_fig-demo.png")
>>> save_fig(png_file_pathname, dpi=600, verbose=True)
```

(continues on next page)

(continued from previous page)

```
Saving "store-save_fig-demo.png" to "tests\images\" ... Done.

>>> svg_file_pathname = cd(img_dir, "store-save_fig-demo.svg")
>>> save_fig(svg_file_pathname, dpi=600, verbose=True, conv_svg_to_emf=True)
Saving "store-save_fig-demo.svg" to "tests\images\" ... Done.
Saving the .svg file as "tests\images\store-save_fig-demo.emf" ... Done.

>>> plt.close()
```

save_web_page_as_pdf

pyhelpers.store.**save_web_page_as_pdf**(*web_page*, *path_to_pdf*, *page_size='A4'*, *zoom=1.0*, *encoding='UTF-8'*, *verbose=False*, *wkhtmltopdf_exe=None*, ***kwargs*)

Save a web page as a PDF file by wkhtmltopdf.

Parameters

- **web_page** (*str*) – URL of a web page or pathname of an HTML file
- **path_to_pdf** (*str*) – path where a .pdf is saved
- **page_size** (*str*) – page size, defaults to `'A4'`
- **zoom** (*float*) – a parameter to zoom in/out, defaults to `1.0`
- **encoding** (*str*) – encoding format defaults to `'UTF-8'`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **wkhtmltopdf_exe** (*str or None*) – absolute path to 'wkhtmltopdf.exe', defaults to `None`; when `wkhtmltopdf_exe=None`, use the default installation path, e.g. (on Windows) *"C:\Program Files\wkhtmltopdf\bin\wkhtmltopdf.exe"*
- **kwargs** – [optional] parameters of pdfkit.from_url

Examples:

```
>>> from pyhelpers.store import save_web_page_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_web_page_as_pdf(web_page_url, pdf_pathname)
>>> # Open the PDF file using the system's default application
>>> subprocess.Popen(pdf_pathname, shell=True)

>>> web_page_file = cd("docs\build\html\index.html")
>>> save_web_page_as_pdf(web_page_file, pdf_pathname, verbose=True)
Updating "pyhelpers.pdf" at "tests\documents\" ... Done.
>>> subprocess.Popen(pdf_pathname, shell=True)
```

(continues on next page)

(continued from previous page)

```
>>> save_web_page_as_pdf(web_page_file, pdf_pathname, verbose=2)
Updating "pyhelpers.pdf" at "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)
```

save_data

pyhelpers.store.**save_data**(*data*, *path_to_file*, *err_warning=True*, *confirmation_required=True*, ***kwargs*)

Save data to a file of a specific format.

Parameters

- **data** (*any*) – data that could be saved to a file of Pickle, CSV, Microsoft Excel, JSON, Joblib or Feather format; a URL of a web page or an HTML file; or an image file of a *Matplotlib*-supported format
- **path_to_file** (*str or os.PathLike[str]*) – pathname of a file that stores the `data`
- **err_warning** (*bool*) – whether to show a warning message if any unknown error occurs, defaults to `True`
- **confirmation_required** (*bool*) – whether to require users to confirm and proceed, defaults to `True`
- **kwargs** – [optional] parameters of one of the following functions: *save_pickle()*, *save_spreadsheet()*, *save_json()*, *save_joblib()*, *save_feather()*, *save_fig()* 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
>>> 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
```

(continues on next page)

(continued from previous page)

```
>>> # Save the data to files different formats:

>>> dat_pathname = cd(data_dir, "dat.pickle")
>>> save_data(dat, dat_pathname, verbose=True)
Saving "dat.pickle" to "tests\data\" ... Done.

>>> dat_pathname = cd(data_dir, "dat.csv")
>>> save_data(dat, dat_pathname, index=True, verbose=True)
Saving "dat.csv" to "tests\data\" ... Done.

>>> dat_pathname = cd(data_dir, "dat.xlsx")
>>> save_data(dat, dat_pathname, index=True, verbose=True)
Saving "dat.xlsx" to "tests\data\" ... Done.

>>> dat_pathname = cd(data_dir, "dat.txt")
>>> save_data(dat, dat_pathname, index=True, verbose=True)
Saving "dat.txt" to "tests\data\" ... Done.

>>> dat_pathname = cd(data_dir, "dat.feather")
>>> save_data(dat, dat_pathname, index=True, verbose=True)
Saving "dat.feather" to "tests\data\" ... Done.

>>> # Convert `dat` to JSON format
>>> import json

>>> dat_ = json.loads(dat.to_json(orient='index'))
>>> 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.json")
>>> save_data(dat_, dat_pathname, indent=4, verbose=True)
Saving "dat.json" to "tests\data\" ... Done.
```

See also:

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

3.4.2 Data loading

load_pickle(path_to_pickle[, verbose])	Load data from a Pickle file.
load_csv(path_to_csv[, delimiter, header, ...])	Load data from a CSV file.
load_spreadsheets(path_to_spreadsheet[, ...])	Load multiple sheets of an Microsoft Excel file.
load_json(path_to_json[, engine, verbose])	Load data from a JSON file.
load_joblib(path_to_joblib[, verbose])	Load data from a joblib file.
load_feather(path_to_feather[, verbose, index])	Load a dataframe from a Feather file.
load_data(path_to_file[, err_warning])	Load data from a file.

load_pickle

pyhelpers.store.**load_pickle**(*path_to_pickle*, *verbose=False*, ***kwargs*)

Load data from a Pickle file.

Parameters

- **path_to_pickle** (*str or os.PathLike[str]*) – path where a pickle file is saved
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of pickle.load

Returns

data retrieved from the specified path `path_to_pickle`

Return type

any

Note:

- Example data can be referred to the function *pyhelpers.store.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_csv*, *delimiter=','*, *header=0*, *index=None*, *verbose=False*, ***kwargs*)

Load data from a CSV file.

Parameters

- **path_to_csv** (*str or os.PathLike[str]*) – path where a CSV file is saved
- **delimiter** (*str*) – delimiter used between values in the data file, defaults to `','`

- **header** (*int or List[int] or None*) – index number of the rows used as column names, defaults to `0`
- **index** (*str or int or list or None*) – index number of the column(s) to use as the row labels of the dataframe, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of csv.reader() or pandas.read_csv()

Returns

data retrieved from the specified path `path_to_csv`

Return type

pandas.DataFrame or None

Note:

- Example data can be referred to the function *pyhelpers.store.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=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

>>> csv_pathname = cd("tests\data", "dat.txt")
>>> csv_dat = load_csv(csv_pathname, index=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_spreadsheet*, *as_dict=True*, *verbose=False*, ***kwargs*)

Load multiple sheets of an Microsoft Excel file.

Parameters

- **path_to_spreadsheet** (*str or os.PathLike[str]*) – path where a spreadsheet is saved
- **as_dict** (*bool*) – whether to return the retrieved data as a dictionary type, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of pandas.ExcelFile.parse

Returns

all worksheet in an Excel workbook from the specified file path `path_to_spreadsheet`

Return type

list or dict

Note:

- Example data can be referred to the functions `pyhelpers.store.save_multiple_spreadsheets()` and *pyhelpers.store.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.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.
>>> list(wb_data.keys())
['TestSheet1', 'TestSheet2', 'TestSheet11', 'TestSheet21']

>>> wb_data = load_spreadsheets(path_to_xlsx, as_dict=False, index_col=0)
>>> type(wb_data)
list
>>> len(wb_data)
4
>>> 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_json*, *engine=None*, *verbose=False*, ***kwargs*)

Load data from a JSON file.

Parameters

- **path_to_json** (*str or os.PathLike[str]*) – path where a json file is saved
- **engine** (*str or None*) – an open-source Python package for JSON serialization, options include `None` (default, for the built-in json module), `'ujson'` (for UltraJSON), `'orjson'` (for orjson) and `'rapidjson'` (for python-rapidjson)
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of 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_json`

Return type

dict

Note:

- Example data can be referred to the function *pyhelpers.store.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.
>>> 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_joblib*, *verbose=False*, ***kwargs*)

Load data from a joblib file.

Parameters

- **path_to_joblib** (*str or os.PathLike[str]*) – path where a joblib file is saved
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **kwargs** – [optional] parameters of joblib.load

Returns

data retrieved from the specified path path_to_joblib

Return type

any

Note:

- Example data can be referred to the function *pyhelpers.store.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],
       ...,
       [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_feather*, *verbose=False*, *index=None*, ***kwargs*)

Load a dataframe from a Feather file.

Parameters

- **path_to_feather** (*str or os.PathLike[str]*) – path where a feather file is saved
- **index** (*str or int or list or None*) – index number of the column(s) to use as the row labels of the dataframe, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **kwargs** – [optional] parameters of pandas.read_feather
  - columns: a sequence of column names, if None, all columns
  - use_threads: whether to parallelize reading using multiple threads, defaults to True

Returns

data retrieved from the specified path path_to_feather

Return type

pandas.DataFrame

Note:

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

Examples:

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

>>> feather_path = cd("tests\data", "dat.feather")
>>> feather_dat = load_feather(feather_path, index=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_data

pyhelpers.store.**load_data**(*path_to_file*, *err_warning=True*, ***kwargs*)

Load data from a file.

Parameters

- **path_to_file** (*str or os.PathLike[str]*) – pathname of a file; supported file formats include Pickle, CSV, Microsoft Excel spreadsheet, JSON, Joblib and Feather
- **err_warning** (*bool*) – whether to show a warning message if any unknown error occurs, defaults to `True`
- **kwargs** – [optional] parameters of one of the following functions: *load_pickle()*, *load_csv()*, load_multiple_spreadsheets(), *load_json()*, *load_joblib()* or *load_feather()*

Returns

loaded data

Return type

any

Note:

- Example data can be referred to the function *pyhelpers.store.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
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=0, verbose=True)
Loading "tests\data\dat.csv" ... Done.
>>> csv_dat
            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

>>> 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=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
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,
        0.1419334 ],
       [0.88498232, 0.19701397, 0.56861333, ..., 0.75842952, 0.02378743,
        0.81357508]])
```

3.4.3 File decompression

unzip(path_to_zip_file[, out_dir, verbose])	Extract data from a zipped (compressed) file.
seven_zip(path_to_zip_file[, out_dir, mode, ...])	Extract data from a compressed file by using 7-Zip.

unzip

pyhelpers.store.**unzip**(*path_to_zip_file*, *out_dir=None*, *verbose=False*, ***kwargs*)

Extract data from a zipped (compressed) file.

Parameters

- **path_to_zip_file** (*str*) – path where a Zip file is saved
- **out_dir** (*str or None*) – path to a directory where the extracted data is saved, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **kwargs** – [optional] parameters of 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.
>>> out_file_pathname = cd("tests\data\zipped", "zipped.txt")
>>> with open(out_file_pathname) as f:
...     print(f.read())
test

>>> output_dir = cd("tests\data\zipped_alt")
>>> unzip(path_to_zip_file=zip_file_path, out_dir=output_dir, verbose=True)
Extracting "tests\data\zipped.zip" to "tests\data\zipped_alt\" ... Done.
>>> out_file_pathname = cd("tests\data\zipped_alt", "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([cd("tests\data\zipped"), output_dir], verbose=True)
To delete the following directories:
    "tests\data\zipped\" (Not empty)
    "tests\data\zipped_alt\" (Not empty)
? [No]|Yes: yes
Deleting "tests\data\zipped\" ... Done.
Deleting "tests\data\zipped_alt\" ... Done.
```

seven_zip

pyhelpers.store.**seven_zip**(*path_to_zip_file*, *out_dir=None*, *mode='aoa'*, *verbose=False*, *seven_zip_exe=None*, ***kwargs*)

Extract data from a compressed file by using 7-Zip.

Parameters

- **path_to_zip_file** (*str*) – path where a compressed file is saved
- **out_dir** (*str or None*) – path to a directory where the extracted data is saved, defaults to None
- **mode** (*str*) – defaults to 'aoa'
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **seven_zip_exe** (*str or None*) – absolute path to '7z.exe', defaults to None; when seven_zip_exe=None, use the default installation path, e.g. (on Windows) *"C:\Program Files\7-Zip\7z.exe"*
- **kwargs** – [optional] parameters of subprocess.run

Examples:

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

>>> zip_file_pathname = cd("tests\data", "zipped.zip")

>>> seven_zip(path_to_zip_file=zip_file_pathname, verbose=True)
7-Zip 20.00 alpha (x64) : Copyright (c) 1999-2020 Igor Pavlov : 2020-02-06

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.

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

>>> output_dir = cd("tests\data\zipped_alt")
>>> seven_zip(path_to_zip_file=zip_file_pathname, out_dir=output_dir, verbose=False)
```

(continues on next page)

(continued from previous page)

```
>>> 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(path_to_zip_file=zip_file_path, out_dir=output_dir)

>>> 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([cd("tests\data\zipped"), output_dir], verbose=True)
To delete the following directories:
    "tests\data\zipped\" (Not empty)
    "tests\data\zipped_alt\" (Not empty)
? [No]|Yes: yes
Deleting "tests\data\zipped\" ... Done.
Deleting "tests\data\zipped_alt\" ... Done.
```

3.4.4 File conversion

markdown_to_rst(path_to_md, path_to_rst[, ...])	Convert a Markdown file (.md) to a reStructuredText (.rst) file.
xlsx_to_csv(xlsx_pathname[, csv_pathname, ...])	Convert Microsoft Excel spreadsheet (in the format .xlsx/.xls) to a CSV file.

markdown_to_rst

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

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

This function relies on Pandoc or pypandoc.

Parameters

- **path_to_md** (*str*) – path where a markdown file is saved
- **path_to_rst** (*str*) – path where a reStructuredText file is saved
- **engine** (*None or str*) – engine/module used for performing the conversion, defaults to None; an alternative option is 'pypandoc'
- **pandoc_exe** (*str or None*) – absolute path to the executable "pandoc.exe", defaults to None; when pandoc_exe=None, use the default installation path, e.g. (on Windows) "*C:\Program Files\Pandoc\pandoc.exe*"

- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of subprocess.run (when `engine=None`) or pypandoc.convert_file (when `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\data\markdown.md" to "tests\data\markdown.rst" ... Done.
```

xlsx_to_csv

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

Convert Microsoft Excel spreadsheet (in the format .xlsx/.xls) to a CSV file.

See also [STORE-XTC-1].

Parameters

- **xlsx_pathname** (*str*) – pathname of an Excel spreadsheet (in the format of .xlsx)
- **csv_pathname** (*str or None*) – pathname of a CSV format file; when `csv_pathname=None` (default), the target CSV file is generated as a tempfile.NamedTemporaryFile; when `csv_pathname=""`, the target CSV file is generated at the same directory where the source Excel spreadsheet is; otherwise, it could also be a specific pathname
- **engine** (*str or None*) – engine used for converting .xlsx/.xls to .csv; when `engine=None` (default), a Microsoft VBScript (Visual Basic Script) is used; when `engine='xlsx2csv'`, the function would rely on xlsx2csv
- **if_exists** (*str*) – how to proceed if the target `csv_pathname` exists, defaults to `'replace'`
- **vbscript** (*str or None*) – pathname of a VB script used for converting .xlsx/.xls to .csv, defaults to `None`
- **sheet_name** (*str*) – name of the target worksheet in the given Excel file, defaults to `'1'`
- **ret_null** – whether to return something depending on the specified `engine`, defaults to `False`

- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of the function subprocess.run

Returns

the pathname of the generated CSV file or None, when `engine=None`; io.StringIO buffer, when `engine='xlsx2csv'`

Return type

str or _io.StringIO or 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=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=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

>>> # Remove the temporary CSV file
>>> os.remove(path_to_temp_csv)
```

3.5 geom

Manipulation of geometric/geographical data.

3.5.1 Geometric data transformation

Geometric type

transform_geom_point_type(*pts[, as_geom])	Transform iterable to shapely.geometry.Point type, or the other way round.

transform_geom_point_type

pyhelpers.geom.**transform_geom_point_type**(**pts*, *as_geom=True*)

Transform iterable to shapely.geometry.Point type, or the other way round.

Parameters

- **pts** (*list or tuple or shapely.geometry.Point*) – data of points (e.g. list of lists/tuples)
- **as_geom** (*bool*) – whether to return point(s) as shapely.geometry.Point, defaults to `True`

Returns

a sequence of points (incl. `None` if errors occur)

Return type

Generator[shapely.geometry.Point, list, tuple, numpy.ndarray]

Examples:

```
>>> from pyhelpers.geom import transform_geom_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_geom_point_type(pt1, pt2)
>>> for x in geom_points:
...     print(x)
```

(continues on next page)

(continued from previous page)

```
POINT (-0.1276474 51.5073219)
POINT (-1.9026911 52.4796992)

>>> geom_points = transform_geom_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(Point, (pt1, pt2))

>>> geom_points = transform_geom_point_type(pt1, pt2)
>>> for x in geom_points:
...     print(x)
POINT (-0.1276474 51.5073219)
POINT (-1.9026911 52.4796992)

>>> geom_points = transform_geom_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_geom_point_type(Point([1, 2, 3]), as_geom=False)
>>> for x in geom_points_:
...     print(x)
(1.0, 2.0, 3.0)
```

Coordinate system

wgs84_to_osgb36(longitudes, latitudes[, ...])	Convert latitude and longitude (WGS84) to British national grid (OSGB36).
osgb36_to_wgs84(eastings, northings[, as_array])	Convert British national grid (OSGB36) to latitude and longitude (WGS84).

wgs84_to_osgb36

pyhelpers.geom.**wgs84_to_osgb36**(*longitudes*, *latitudes*, *as_array=False*, ***kwargs*)

Convert latitude and longitude (WGS84) to British national grid (OSGB36).

Parameters

- **longitudes** (*int or float or Iterable[int, float]*) – the longitude (abbr: long., λ, or lambda) of a point on Earth's surface
- **latitudes** (*int or float or Iterable[int, float]*) – the latitude (abbr: lat., φ, or phi) of a point on Earth's surface
- **as_array** (*bool*) – whether to return an array, defaults to `False`
- **kwargs** – [optional] parameters of pyproj.Transformer.transform

Returns
geographic Cartesian coordinate (Easting, Northing) or (X, Y)

Return type
tuple or 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*)

Convert British national grid (OSGB36) to latitude and longitude (WGS84).

Parameters

- **eastings** (*int or float or Iterable[int, float]*) – Easting (X), eastward-measured distance (or the x-coordinate)

- **northings** (*int or float or Iterable[int, float]*) – Northing (Y), northward-measured distance (or the y-coordinate)
- **as_array** (*bool*) – whether to return an array, defaults to `False`
- **kwargs** – [optional] parameters of pyproj.Transformer.transform

Returns
geographic coordinate (Longitude, Latitude)

Return type
tuple or numpy.ndarry

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

drop_axis(geom[, axis, as_array])	Drop an axis from a given 3D geometry object.
project_point_to_line(point, line[, ...])	Find the projected point from a known point to a line.

drop_axis

pyhelpers.geom.**drop_axis**(*geom*, *axis='z'*, *as_array=False*)

Drop an axis from a given 3D geometry object.

Parameters

- **geom** (*shapely.geometry object*) – geometry object that has x, y and z coordinates
- **axis** (*str*) – options include 'x', 'y' and 'z', defaults to `'z'`
- **as_array** (*bool*) – whether to return an array, defaults to `False`

Returns

geometry object (or an array) without the specified `axis`

Return type

shapely.geometry object or 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]])
```

(continues on next page)

(continued from previous page)

```
>>> 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*)

Find the projected point from a known point to a line.

Parameters

- **point** (*shapely.geometry.Point*) – geometry object of a point
- **line** (*shapely.geometry.LineString*) – geometry object of a line
- **drop_dimension** (*str or None*) – which dimension to drop, defaults to None; options include 'x', 'y' and 'z'

Returns

the original point (with all or partial dimensions, given drop) and the projected one

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 Fig. 10):

```
>>> import matplotlib.pyplot as plt
>>> from pyhelpers.settings import mpl_preferences

>>> mpl_preferences(font_name='Times New Roman', 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.text(pos[0], pos[1], pos[2], str(pos))

>>> ax.legend(loc=3)
>>> plt.tight_layout()

>>> ax.set_xticklabels([])
>>> ax.set_yticklabels([])
>>> ax.set_zticklabels([])

>>> plt.show()
```

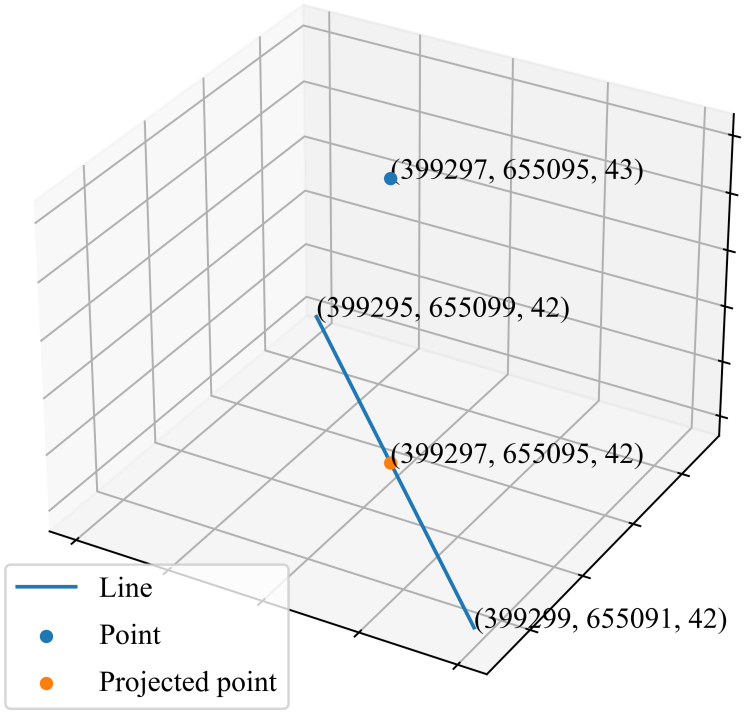

Fig. 10: An example of projecting a point onto a line.

3.5.2 Geometric data computation

Distance

calc_distance_on_unit_sphere(pt1, pt2[, verbose])	Calculate distance between two points.
calc_hypotenuse_distance(pt1, pt2)	Calculate hypotenuse given two points (the right-angled triangle, given its side and perpendicular).
find_closest_point(pt, ref_pts[, as_geom])	Find the closest point of the given point to a list of points.
find_closest_points(pts, ref_pts[, k, ...])	Find the closest points from a list of reference points (applicable for vectorized computation).
find_shortest_path(points_sequence[, ...])	Find the shortest path through a sequence of points.

calc_distance_on_unit_sphere

pyhelpers.geom.**calc_distance_on_unit_sphere**(*pt1*, *pt2*, *verbose=False*)

Calculate distance between two points.

Parameters

- **pt1** (*shapely.geometry.Point or list or tuple or numpy.ndarray*) – a point
- **pt2** (*shapely.geometry.Point or list or tuple or numpy.ndarray*) – another point
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns

distance (in miles) between `pt1` and `pt2` (relative to the earth's radius)

Return type

float

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.

Examples:

```
>>> from pyhelpers.geom import calc_distance_on_unit_sphere
>>> 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
>>> arc_len_in_miles = calc_distance_on_unit_sphere(london, birmingham)
>>> arc_len_in_miles
101.10431101941569
```

calc_hypotenuse_distance

pyhelpers.geom.**calc_hypotenuse_distance**(*pt1*, *pt2*)

Calculate hypotenuse given two points (the right-angled triangle, given its side and perpendicular).

See also [GEOM-CHD-1].

Parameters

- **pt1** (*shapely.geometry.Point or list or tuple or numpy.ndarray*) – a point
- **pt2** (*shapely.geometry.Point or list or tuple or numpy.ndarray*) – another point

Returns
hypotenuse

Return type
float

Note:

- This is the length of the vector from the `orig_pt` to `dest_pt`.
- `numpy.hypot(x, y)` return the Euclidean norm, `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(Point, (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*)

Find the closest point of the given point to a list of points.

Parameters

- **pt** (*tuple or list or shapely.geometry.Point*) – (longitude, latitude)
- **ref_pts** (*Iterable or numpy.ndarray or list or tuple or shapely.geometry.Point or shapely.geometry.MultiPoint or shapely.geometry.LineString or shapely.geometry.MultiLineString or shapely.geometry.Polygon or shapely.geometry.MultiPolygon or shapely.geometry.GeometryCollection*) – a sequence of reference (tuple/list of length 2) points
- **as_geom** (*bool*) – whether to return shapely.geometry.Point, defaults to `True`

Returns

the point closest to `pt`

Return type

shapely.geometry.Point or 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
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*)

Find the closest points from a list of reference points (applicable for vectorized computation).

See also [GEOM-FCPB-1].

Parameters

- **pts** (*numpy.ndarray or list or tuple or Iterable or shapely.geometry.Point or shapely.geometry.MultiPoint or shapely.geometry.LineString or shapely.geometry.MultiLineString or shapely.geometry.Polygon or shapely.geometry.MultiPolygon or shapely.geometry.GeometryCollection*) – an array (of size (n, 2)) of points
- **ref_pts** (*numpy.ndarray or list or tuple or shapely.geometry.Point or shapely.geometry.MultiPoint or shapely.geometry.LineString or shapely.geometry.MultiLineString or shapely.geometry.Polygon or shapely.geometry.MultiPolygon or shapely.geometry.GeometryCollection*) – an array (of size (n, 2)) of reference points
- **k** (*int or list*) – (up to) the k-th nearest neighbour(s), defaults to `1`
- **unique** (*bool*) – whether to remove duplicated points, defaults to `False`
- **as_geom** (*bool*) – whether to return 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] parameters of scipy.spatial.cKDTree

Returns

point (or points) among the list of `ref_pts`, which is (or are) closest to `pts`

Return type

numpy.ndarray or 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
```

(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

>>> 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], dtype=int64)

>>> _, _, 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*)

Find the shortest path through a sequence of points.

Parameters

- **points_sequence** (*numpy.ndarray*) – a sequence of points
- **ret_dist** (*bool*) – whether to return the distance of the shortest path, defaults to False
- **as_geom** (*bool*) – whether to return the sorted path as a line geometry object,

defaults to `False`

- **kwargs** – (optional) parameters used by sklearn.neighbors.NearestNeighbors

Returns
a sequence of sorted points given two-nearest neighbors

Return type
numpy.ndarray or shapely.geometry.LineString or tuple

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],
       [-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 Fig. 11):

```
>>> import matplotlib.pyplot as plt
>>> import matplotlib.gridspec as mgs
>>> from pyhelpers.settings import mpl_preferences

>>> mpl_preferences(font_name='Times New Roman')

>>> fig = plt.figure(figsize=(7, 5))
>>> gs = mgs.GridSpec(1, 2, figure=fig)
```

(continues on next page)

(continued from previous page)

```
>>> 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)

>>> plt.tight_layout()
>>> plt.show()
```

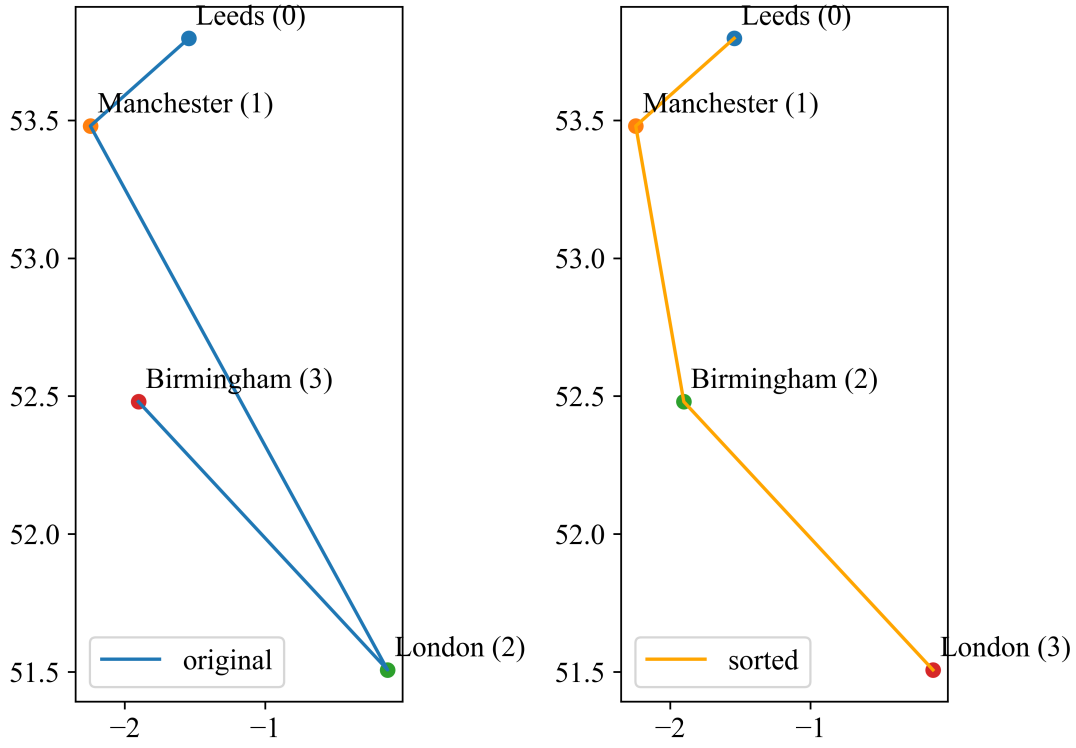

Fig. 11: An example of sorting a sequence of points given the shortest path.

Locating

get_midpoint(x1, y1, x2, y2[, as_geom])	Get the midpoint between two points (applicable for vectorized computation).
get_geometric_midpoint(pt1, pt2[, as_geom])	Get the midpoint between two points.
get_geometric_midpoint_calc(pt1, pt2[, as_geom])	Get the midpoint between two points by pure calculation.
get_rectangle_centroid(rectangle[, as_geom])	Get coordinates of the centroid of a rectangle
get_square_vertices(ctr_x, ctr_y, side_length)	Get the four vertices of a square given its centre and side length.
get_square_vertices_calc(ctr_x, ctr_y, ...)	Get the four vertices of a square given its centre and side length (by elementary calculation).

get_midpoint

pyhelpers.geom.**get_midpoint**(*x1*, *y1*, *x2*, *y2*, *as_geom=False*)

Get the midpoint between two points (applicable for vectorized computation).

Parameters

- **x1** (*float or int or Iterable or numpy.ndarray*) – longitude(s) or easting(s) of a point (an array of points)
- **y1** (*float or int or Iterable or numpy.ndarray*) – latitude(s) or northing(s) of a point (an array of points)
- **x2** (*float or int or Iterable or numpy.ndarray*) – longitude(s) or easting(s) of another point (another array of points)
- **y2** (*float or int or Iterable or 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 or shapely.geometry.Point or 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)
```

(continues on next page)

(continued from previous page)

```
>>> 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*)

Get the midpoint between two points.

Parameters

- **pt1** (*shapely.geometry.Point or list or tuple or numpy.ndarray*) – a point
- **pt2** (*shapely.geometry.Point or list or tuple or numpy.ndarray*) – another point
- **as_geom** (*bool*) – whether to return shapely.geometry.Point, defaults to `False`

Returns

the midpoint between `pt1` and `pt2`

Return type

tuple or shapely.geometry.Point or 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 *pyhelpers.geom.get_geometric_midpoint_calc()*.

get_geometric_midpoint_calc

pyhelpers.geom.**get_geometric_midpoint_calc**(*pt1*, *pt2*, *as_geom=False*)

Get the midpoint between two points by pure calculation.

See also [GEOM-GGMC-1] and [GEOM-GGMC-2].

Parameters

- **pt1** (*shapely.geometry.Point or list or tuple or numpy.ndarray*) – a point
- **pt2** (*shapely.geometry.Point or list or tuple or numpy.ndarray*) – a point
- **as_geom** (*bool*) – whether to return shapely.geometry.Point. defaults to `False`

Returns

the midpoint between `pt1` and `pt2`

Return type

tuple or shapely.geometry.Point or 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)
>>> 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 *pyhelpers.geom.get_geometric_midpoint()*.

get_rectangle_centroid

pyhelpers.geom.**get_rectangle_centroid**(*rectangle*, *as_geom=False*)

Get coordinates of the centroid of a rectangle

Parameters

- **rectangle** (*list or tuple or numpy.ndarray or shapely.geometry.Polygon or shapely.geometry.MultiPolygon*) – polygon or multipolygon geometry object
- **as_geom** (*bool*) – whether to return a shapely.geometry object

Returns

coordinate of the rectangle

Return type

numpy.ndarray or 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]]

>>> rect_obj = Polygon(coords_1)
>>> rect_cen = get_rectangle_centroid(rectangle=rect_obj)
>>> rect_cen
array([0.5, 0.5])

>>> rect_obj = numpy.array(coords_1)
>>> rect_cen = get_rectangle_centroid(rectangle=rect_obj)
>>> rect_cen
array([0.5, 0.5])

>>> rect_cen = get_rectangle_centroid(rectangle=rect_obj, as_geom=True)
>>> type(rect_cen)
shapely.geometry.point.Point
>>> rect_cen.wkt
'POINT (0.5 0.5)'

>>> coords_2 = [[(0, 0), (0, 1), (1, 1), (1, 0)], [(1, 1), (1, 2), (2, 2), (2, 1)]]

>>> rect_cen = get_rectangle_centroid(rectangle=coords_2)
```

get_square_vertices

pyhelpers.geom.**get_square_vertices**(*ctr_x*, *ctr_y*, *side_length*, *rotation_theta=0*)

Get the four vertices of a square given its centre and side length.

See also [GEOM-GSV-1].

Parameters

- **ctr_x** (*int or float*) – x coordinate of a square centre
- **ctr_y** (*int or float*) – y coordinate of a square centre
- **side_length** (*int or float*) – side length of a square
- **rotation_theta** (*int or float*) – rotate (anticlockwise) the square by `rotation_theta` (in degree), defaults to `0`

Returns

vertices of the square as an array([ll, ul, ur, lr])

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*)

Get the four vertices of a square given its centre and side length (by elementary calculation).

See also [GEOM-GSVC-1].

Parameters

- **ctr_x** (*int or float*) – x coordinate of a square centre

- **ctr_y** (*int or float*) – y coordinate of a square centre
- **side_length** (*int or float*) – side length of a square
- **rotation_theta** (*int or float*) – rotate (anticlockwise) the square by `rotation_theta` (in degree), defaults to `0`

Returns
vertices of the square as an array([ll, ul, ur, lr])

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 *pyhelpers.geom.get_square_vertices()*.

3.5.3 Geometric data sketching

sketch_square(ctr_x, ctr_y, side_length[, ...])	Sketch a square given its centre point, four vertices and rotation angle (in degree).

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*)

Sketch a square given its centre point, four vertices and rotation angle (in degree).

Parameters

- **ctr_x** (*int or float*) – x coordinate of a square centre
- **ctr_y** (*int or float*) – y coordinate of a square centre
- **side_length** (*int or float*) – side length of a square
- **rotation_theta** (*int or float*) – rotate (anticlockwise) the square by `rotation_theta` (in degree), defaults to `0`
- **annotation** (*bool*) – whether to annotate vertices of the square, defaults to `True`
- **annot_font_size** (*int*) – font size annotation texts, defaults to `12`
- **fig_size** (*tuple or list*) – figure size, defaults to `(6.4, 4.8)`
- **ret_vertices** (*bool*) – whether to return the vertices of the square, defaults to `False`
- **kwargs** – [optional] parameters of 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()
```

The above exmaple is illustrated in Fig. 12:

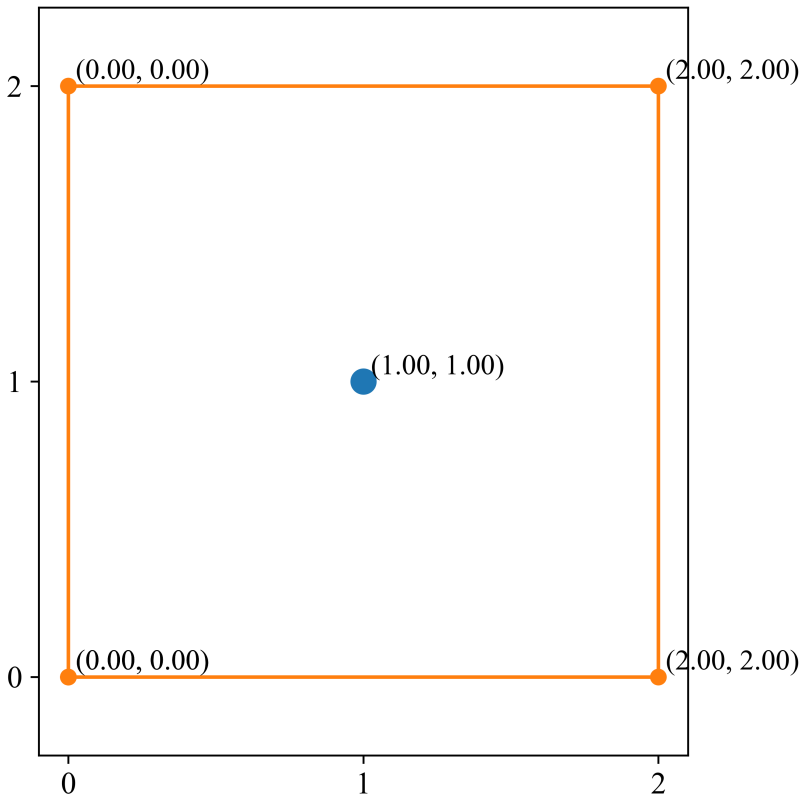

Fig. 12: An example of a sketch of a square, created by the function *sketch_square()*.

```
>>> sketch_square(c1, c2, side_len, rotation_theta=75, annotation=True, fig_size=(5, 5))
>>> plt.show()
```

This second example is illustrated in Fig. 13:

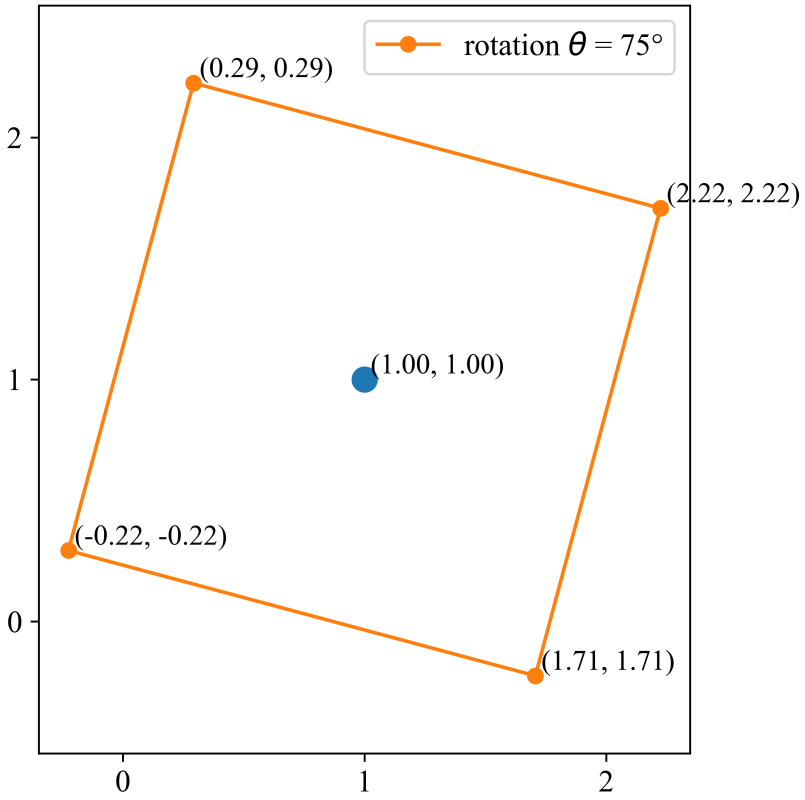

Fig. 13: An example of a sketch of a square rotated 75 degrees anticlockwise about the centre.

3.6 text

Manipulation of textual data.

3.6.1 Textual data preprocessing

get_acronym(text[, only_capitals, ...])	Get an acronym (in capital letters) of an input text.
remove_punctuation(x[, rm_whitespace])	Remove punctuation from string-type data.
extract_words1upper(x[, join_with])	Extract words from a string by spliting it at occurrence of an uppercase letter.
numeral_english_to_arabic(x)	Convert a string which potentially is a number written in English to an Arabic number

get_acronym

pyhelpers.text.**get_acronym**(*text*, *only_capitals=False*, *capitals_in_words=False*)

Get an acronym (in capital letters) of an input text.

Parameters

- **text** (*str*) – any text
- **only_capitals** (*bool*) – whether to include capital letters only, defaults to `False`
- **capitals_in_words** (*bool*) – whether to include all captical letters in a single word, defaults to `False`

Returns

acronym of the input `str_var`

Return type

str

Examples:

```
>>> from pyhelpers.text import get_acronym

>>> text_a = 'This is an apple.'
>>> acron = get_acronym(text_a)
>>> acron
'TIAA'

>>> text_b = "I'm at the University of Birmingham."
>>> acron = get_acronym(text_b, only_capitals=True)
>>> acron
'IUB'

>>> text_c = 'There is a "ConnectionError"!'
>>> acron = get_acronym(text_c, capitals_in_words=True)
>>> acron
'TCE'
```

remove_punctuation

pyhelpers.text.**remove_punctuation**(*x*, *rm_whitespace=True*)

Remove punctuation from string-type data.

Parameters

- **x** (*str*) – raw string-type data
- **rm_whitespace** (*bool*) – whether to remove whitespace (incl. escape characters), defaults to `True`

Returns

text with punctuation removed

Return type
str

Examples:

```
>>> from pyhelpers.text import remove_punctuation

>>> raw_text = 'Hello world!    This is a test. :-)'

>>> text = remove_punctuation(raw_text)
>>> text
'Hello world This is a test'

>>> text = remove_punctuation(raw_text, rm_whitespace=False)
>>> text
'Hello world    This is a test'
```

extract_words1upper

pyhelpers.text.**extract_words1upper**(*x*, *join_with=None*)

Extract words from a string by spliting it at occurrence of an uppercase letter.

Parameters

- **x** (*str*) – a string joined by a number of words each starting with an uppercase letter
- **join_with** (*str or None*) – a string with which to (re)join the single words, defaults to None

Returns
a list of single words each starting with an uppercase letter, or a single string joined together by them with join_with

Return type
list or str

Examples:

```
>>> from pyhelpers.text import extract_words1upper

>>> x1 = 'Network_Waymarks'
>>> x1_ = extract_words1upper(x1)
>>> x1_
['Network', 'Waymarks']

>>> x2 = 'NetworkRailRetainingWall'
>>> x2_ = extract_words1upper(x2, join_with=' ')
>>> x2_
'Network Rail Retaining Wall'
```

numeral_english_to_arabic

pyhelpers.text.**numeral_english_to_arabic**(*x*)

Convert a string which potentially is a number written in English to an Arabic number

Parameters
x (*str*) – a number written in English

Returns
a number written in Arabic

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
Exception: Illegal word: "fivety"
```

3.6.2 Textual data computation

count_words(raw_txt)	Count the total for each different word.
calculate_idf(raw_documents[, rm_punc])	Calculate inverse document frequency.
calculate_tf_idf(raw_documents[, rm_punc])	Count term frequency–inverse document frequency.
euclidean_distance_between_texts(txt1, txt2)	Compute Euclidean distance of two sentences.
cosine_similarity_between_texts(txt1, txt2)	Calculate cosine similarity of two sentences.

count_words

pyhelpers.text.**count_words**(*raw_txt*)

Count the total for each different word.

Parameters
raw_txt (*str*) – any text

Returns
number of each word in raw_docs

Return type
dict

Examples:

```
>>> from pyhelpers.text import count_words, remove_punctuation

>>> raw_text = 'This is an apple. That is a pear. Hello world!'

>>> count_words(raw_text)
{'This': 1,
 'is': 2,
 'an': 1,
 'apple': 1,
 '.': 2,
 'That': 1,
 'a': 1,
 'pear': 1,
 'Hello': 1,
 'world': 1,
 '!': 1}

>>> count_words(remove_punctuation(raw_text))
{'This': 1,
 'is': 2,
 'an': 1,
 'apple': 1,
 'That': 1,
 'a': 1,
 'pear': 1,
 'Hello': 1,
 'world': 1}
```

calculate_idf

pyhelpers.text.**calculate_idf**(*raw_documents*, *rm_punc=False*)

Calculate inverse document frequency.

Parameters

- **raw_documents** (*Iterable or Sequence*) – a sequence of textual data
- **rm_punc** (*bool*) – whether to remove punctuation from the input textual data, defaults to False

Returns

term frequency (TF) of the input textual data, and inverse document frequency

Return type

tuple[list[dict], dict]

Examples:

```
>>> from pyhelpers.text import calculate_idf

>>> raw_doc = [
...     'This is an apple.',
...     'That is a pear.',
...     'It is human being.',
...     'Hello world!']

>>> docs_tf_, corpus_idf_ = calculate_idf(raw_doc, rm_punc=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}

>>> docs_tf_, corpus_idf_ = calculate_idf(raw_doc, rm_punc=True)
>>> 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,
```

(continues on next page)

(continued from previous page)

```
 'world': 0.6931471805599453}
```

calculate_tf_idf

pyhelpers.text.**calculate_tf_idf**(*raw_documents*, *rm_punc=False*)

Count term frequency–inverse document frequency.

Parameters

- **raw_documents** (*Iterable or Sequence*) – a sequence of textual data
- **rm_punc** (*bool*) – whether to remove punctuation from the input textual data, defaults to `False`

Returns

tf-idf of the input textual data

Return type

dict

Examples:

```
>>> from pyhelpers.text import calculate_tf_idf

>>> raw_doc = [
...     'This is an apple.',
...     'That is a pear.',
...     'It is human being.',
...     'Hello world!']

>>> docs_tf_idf_ = calculate_tf_idf(raw_documents=raw_doc)
>>> docs_tf_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}

>>> docs_tf_idf_ = calculate_tf_idf(raw_documents=raw_doc, rm_punc=True)
>>> docs_tf_idf_
{'This': 0.6931471805599453,
 'is': 0.0,
 'an': 0.6931471805599453,
 'apple': 0.6931471805599453,
 'That': 0.6931471805599453,
```

(continues on next page)

(continued from previous page)

```
'a': 0.6931471805599453,
'pear': 0.6931471805599453,
'It': 0.6931471805599453,
'human': 0.6931471805599453,
'being': 0.6931471805599453,
'Hello': 0.6931471805599453,
'world': 0.6931471805599453}
```

euclidean_distance_between_texts

pyhelpers.text.**euclidean_distance_between_texts**(*txt1*, *txt2*)

Compute Euclidean distance of two sentences.

Parameters

- **txt1** (*str*) – any text
- **txt2** (*str*) – any text

Returns

Euclidean distance between the input textual data

Return type

float

Examples:

```
>>> from pyhelpers.text import euclidean_distance_between_texts

>>> txt_1, txt_2 = 'This is an apple.', 'That is a pear.'

>>> euclidean_distance = euclidean_distance_between_texts(txt_1, txt_2)
>>> euclidean_distance
2.449489742783178
```

cosine_similarity_between_texts

pyhelpers.text.**cosine_similarity_between_texts**(*txt1*, *txt2*, *cosine_distance=False*)

Calculate cosine similarity of two sentences.

Parameters

- **txt1** (*str*) – any text
- **txt2** (*str*) – any text
- **cosine_distance** (*bool*) – whether to get cosine distance, which is (1 - cosine similarity), defaults to `False`

Returns

cosine similarity (or distance)

Return type
float

Examples:

```
>>> from pyhelpers.text import cosine_similarity_between_texts

>>> txt_1, txt_2 = 'This is an apple.', 'That is a pear.'

>>> cos_sim = cosine_similarity_between_texts(txt_1, txt_2)
>>> cos_sim
0.25

>>> cos_dist = cosine_similarity_between_texts(txt_1, txt_2, cosine_distance=True)
>>> cos_dist  # 1 - cos_sim
0.75
```

3.6.3 Textual data comparison

find_matched_str(x, lookup_list)	Find all that are matched with a string from among a sequence of strings.
find_similar_str(x, lookup_list[, n, ...])	Find `n` strings that are similar to `x` from among a sequence of candidates.

find_matched_str

pyhelpers.text.**find_matched_str**(*x*, *lookup_list*)

Find all that are matched with a string from among a sequence of strings.

Parameters

- **x** (*str*) – a string-type variable
- **lookup_list** (*Iterable*) – a sequence of strings for lookup

Returns
a generator containing all that are matched with `x`

Return type
Generator or None

Examples:

```
>>> from pyhelpers.text import find_matched_str

>>> lookup_lst = ['abc', 'aapl', 'app', 'ap', 'ape', 'apex', 'apel']
>>> res = find_matched_str('apple', lookup_lst)
>>> list(res)
[]

>>> lookup_lst += ['apple']
>>> lookup_lst
```

(continues on next page)

(continued from previous page)

```
['abc', 'aapl', 'app', 'ap', 'ape', 'apex', 'apel', 'apple']

>>> res = find_matched_str('apple', lookup_lst)
>>> list(res)
['apple']

>>> res = find_matched_str(r'app(le)?', lookup_lst)
>>> list(res)
['app', 'apple']
```

find_similar_str

pyhelpers.text.**find_similar_str**(*x*, *lookup_list*, *n=1*, *ignore_punctuation=True*, *engine='difflib'*, ***kwargs*)

Find `n` strings that are similar to `x` from among a sequence of candidates.

Parameters

- **x** (*str*) – a string-type variable
- **lookup_list** (*Iterable*) – a sequence of strings for lookup
- **n** (*int or None*) – number of similar strings to return, defaults to `1`; when `n=None`, the function returns a sorted `lookup_list` (in the descending order of similarity)
- **ignore_punctuation** (*bool*) – whether to ignore punctuations in the search for similar texts, defaults to `True`
- **engine** (*str or Callable*) – options include `'difflib'` (default) and `'fuzzywuzzy'`
  - if `engine='difflib'`, the function relies on difflib.get_close_matches
  - if `engine='fuzzywuzzy'`, the function relies on fuzzywuzzy.fuzz.token_set_ratio
- **kwargs** – [optional] parameters of difflib.get_close_matches (e.g. `cutoff=0.6`) or fuzzywuzzy.fuzz.token_set_ratio, depending on `engine`

Returns

a string-type variable that should be similar to (or the same as) `x`

Return type

str or list or None

Note:

- By default, the function uses the built-in module difflib; when we set the parameter `engine='fuzzywuzzy'`, the function then relies on FuzzyWuzzy, which is not an essential dependency for installing pyhelpers. We could however use `pip` (or `conda`) to install it first separately.

Examples:

```
>>> from pyhelpers.text import find_similar_str

>>> lookup_lst = ['Anglia',
...               'East Coast',
...               'East Midlands',
...               'North and East',
...               'London North Western',
...               'Scotland',
...               'South East',
...               'Wales',
...               'Wessex',
...               'Western']

>>> y = find_similar_str(x='angle', lookup_list=lookup_lst)
>>> y
'Anglia'
>>> y = find_similar_str(x='angle', lookup_list=lookup_lst, n=2)
>>> y
['Anglia', 'Wales']

>>> y = find_similar_str(x='angle', lookup_list=lookup_lst, engine='fuzzywuzzy')
>>> y
'Anglia'
>>> y = find_similar_str('angle', lookup_lst, n=2, engine='fuzzywuzzy')
>>> y
['Anglia', 'Wales']

>>> y = find_similar_str(x='x', lookup_list=lookup_lst)
>>> y is None
True
>>> y = find_similar_str(x='x', lookup_list=lookup_lst, cutoff=0.25)
>>> y
'Wessex'
>>> y = find_similar_str(x='x', lookup_list=lookup_lst, n=2, cutoff=0.25)
>>> y
'Wessex'

>>> y = find_similar_str(x='x', lookup_list=lookup_lst, engine='fuzzywuzzy')
>>> y
'Wessex'
>>> y = find_similar_str(x='x', lookup_list=lookup_lst, n=2, engine='fuzzywuzzy')
>>> y
['Wessex', 'Western']
```

3.7 dbms

Communication with databases.

The current release includes classes for PostgreSQL *and* Microsoft SQL Server.

3.7.1 Databases

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=True*)

A class for basic communication with PostgreSQL databases.

Parameters

- **host** (*str or None*) – host name/address of a PostgreSQL server, e.g. `'localhost'` or `'127.0.0.1'` (default by installation of PostgreSQL); when `host=None` (default), it is initialized as `'localhost'`
- **port** (*int or None*) – listening port used by PostgreSQL; when `port=None` (default), it is initialized as `5432` (default by installation of PostgreSQL)
- **username** (*str or None*) – username of a PostgreSQL server; when `username=None` (default), it is initialized as `'postgres'` (default by installation of PostgreSQL)
- **password** (*str or int or None*) – user password; when `password=None` (default), it is required to mannually type in the correct password to connect the PostgreSQL server
- **database_name** (*str or None*) – name of a database; when `database=None` (default), it is initialized as `'postgres'` (default by installation of PostgreSQL)
- **confirm_db_creation** (*bool*) – whether to prompt a confirmation before creating a new database (if the specified database does not exist), defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `True`

Variables

- **host** (*str*) – host name/address

- **port** (*str*) – listening port used by PostgreSQL
- **username** (*str*) – username
- **database_name** (*str*) – name of a 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('localhost', 5432, 'postgres', database_name='postgres')
>>> postgres = PostgreSQL()
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')
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')
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

DEFAULT_DATABASE	Name of the database that is by default to connect with.
DEFAULT_DIALECT	Default dialect.
DEFAULT_DRIVER	Default name of database driver.
DEFAULT_HOST	Default host name/address (by installation of PostgreSQL).
DEFAULT_PORT	Default listening port used by PostgreSQL.
DEFAULT_SCHEMA	Name of the schema that is created by default for the installation of PostgreSQL.
DEFAULT_USERNAME	Default username.

PostgreSQL.DEFAULT_DATABASE

PostgreSQL.**DEFAULT_DATABASE = 'postgres'**

Name of the database that is by default to connect with. The database is created by installation of PostgreSQL.

PostgreSQL.DEFAULT_DIALECT

PostgreSQL.**DEFAULT_DIALECT = 'postgresql'**

Default dialect. The dialect that SQLAlchemy uses to communicate with PostgreSQL; see also [DBMS-PS-1]

PostgreSQL.DEFAULT_DRIVER

PostgreSQL.**DEFAULT_DRIVER = 'psycopg2'**

Default name of database driver.

PostgreSQL.DEFAULT_HOST

PostgreSQL.**DEFAULT_HOST = 'localhost'**

Default host name/address (by installation of PostgreSQL).

PostgreSQL.DEFAULT_PORT

PostgreSQL.**DEFAULT_PORT = 5432**

Default listening port used by PostgreSQL.

PostgreSQL.DEFAULT_SCHEMA

PostgreSQL.**DEFAULT_SCHEMA = 'public'**

Name of the schema that is created by default for the installation of PostgreSQL.

PostgreSQL.DEFAULT_USERNAME

PostgreSQL.**DEFAULT_USERNAME = 'postgres'**

Default username.

Methods

add_primary_keys(primary_keys, table_name[, ...])	Add a primary key or multiple primary keys to a table.
alter_table_schema(table_name, schema_name, ...)	Move a table from one schema to another within the currently-connected database.
connect_database([database_name, verbose])	Establish a connection to a database.
create_database(database_name[, verbose])	Create a database.
create_schema(schema_name[, verbose])	Create a schema.
create_table(table_name, column_specs[, ...])	Create a table.
database_exists([database_name])	Check whether a database exists.
disconnect_all_others()	Kill connections to all databases except the currently-connected one.
disconnect_database([database_name, verbose])	Disconnect a database.
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_dtype(table_name[, column_names, ...])	Get information about data types of all or specific columns of a table.
get_column_info(table_name[, schema_name, ...])	Get information about columns of a table.
get_database_names([names_only])	Get names of all existing databases.
get_database_size([database_name])	Get the size of a database.
get_primary_keys(table_name[, schema_name, ...])	Get the primary keys of a table.
get_schema_names([include_all, names_only, ...])	Get the names of existing schemas.
get_table_names([schema_name, verbose])	Get the names of all tables in a schema.
import_data(data, table_name[, schema_name, ...])	Import tabular data into a table.
null_text_to_empty_string(table_name[, ...])	Convert null values (in text columns) to empty strings.
psql_insert_copy(sql_table, sql_db_engine, ...)	A callable using PostgreSQL COPY clause for executing inserting data.
read_sql_query(sql_query[, method, ...])	Read table data by SQL query (recommended for large table).
read_table(table_name[, schema_name, ...])	Read data from a table.
schema_exists(schema_name)	Check whether a schema exists.
table_exists(table_name[, schema_name])	Check whether a table exists.

PostgreSQL.add_primary_keys

PostgreSQL.**add_primary_keys**(*primary_keys*, *table_name*, *schema_name=None*)

Add a primary key or multiple primary keys to a table.

Parameters

- **primary_keys** (*str or list or None*) – (list of) primary key(s)
- **table_name** (*str*) – name of a table
- **schema_name** (*str or None*) – name of a schema, defaults to None

See also:

- Examples for the method *PostgreSQL.get_primary_keys()*.

PostgreSQL.alter_table_schema

PostgreSQL.**alter_table_schema**(*table_name*, *schema_name*, *new_schema_name*, *confirmation_required=True*, *verbose=False*)

Move a table from one schema to another within the currently-connected database.

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str*) – name of a schema
- **new_schema_name** – name of a new schema
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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.
```

(continues on next page)

(continued from previous page)

```
>>> # 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.

>>> lst_tbl_names = testdb.get_table_names(schema_name='test_schema')
>>> lst_tbl_names
['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*)

Establish a connection to a database.

Parameters

- **database_name** (*str or None*) – name of a database; when database_name=None (default), the database name is input manually
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
Password (postgres@localhost:5432): ***
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'
```

(continues on next page)

(continued from previous page)

```
>>> 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*)

Create a database.

Parameters

- **database_name** (*str*) – name of a database
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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
'postgres'
```

PostgreSQL.create_schema

PostgreSQL.**create_schema**(*schema_name*, *verbose=False*)

Create a schema.

Parameters

- **schema_name** (*str*) – name of a schema
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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*)

Create a table.

Parameters

- **table_name** (*str*) – name of a table
- **column_specs** (*str*) – specifications for each column of the table
- **schema_name** (*str or None*) – name of a schema; when schema_name=None (default), it defaults to *DEFAULT_SCHEMA* (i.e. 'public')
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> 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
                            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
column_default                  None          None
is_nullable                      YES           YES
data_type                    integer          text
character_maximum_length        None          None
character_octet_length           NaN  1073741824.0
numeric_precision               32.0           NaN
numeric_precision_radix          2.0           NaN
numeric_scale                    0.0           NaN
datetime_precision              None          None
interval_type                   None          None
interval_precision              None          None
character_set_catalog           None          None
character_set_schema            None          None
character_set_name              None          None
collation_catalog               None          None
collation_schema                None          None
collation_name                  None          None
domain_catalog                  None          None
domain_schema                   None          None
domain_name                     None          None
udt_catalog                   testdb        testdb
udt_schema                pg_catalog    pg_catalog
udt_name                        int4          text
scope_catalog                   None          None
scope_schema                    None          None
scope_name                      None          None
maximum_cardinality             None          None
dtd_identifier                     1             2
is_self_referencing               NO            NO
is_identity                       NO            NO
identity_generation             None          None
identity_start                  None          None
```

(continues on next page)

(continued from previous page)

```
identity_increment              None          None
identity_maximum                None          None
identity_minimum                None          None
identity_cycle                    NO            NO
is_generated                   NEVER         NEVER
generation_expression           None          None
is_updatable                     YES           YES

>>> # Get data types of all columns of the table "public"."test_table"
>>> test_tbl_dtypes = testdb.get_column_dtype(table_name=tbl_name)
>>> test_tbl_dtypes
{'col_name_1': 'integer', 'col_name_2': 'text'}

>>> # 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*)

Check whether a database exists.

Parameters

database_name (*str or None*) – name of a database, defaults to None

Returns

whether the database exists

Return type

bool

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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'
```

(continues on next page)

(continued from previous page)

```
>>> # 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**()

Kill connections to all databases except the currently-connected one.

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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*)

Disconnect a database.

See also [DBMS-PS-DD-1].

Parameters

- **database_name** (*str or None*) – name of database to disconnect from; if database_name=None (default), disconnect the current database.

- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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*)

Delete/drop a database.

Parameters

- **database_name** (*str or None*) – database to be disconnected; if `database_name=None` (default), drop the database being currently currented
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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
```

(continues on next page)

(continued from previous page)

```
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*)

Delete/drop one or multiple schemas.

Parameters

- **schema_names** (*str or Iterable[str]*) – name of one schema, or names of multiple schemas
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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.)
```

(continues on next page)

(continued from previous page)

```
>>> 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*)

Delete/drop a table.

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str or None*) – name of a schema; when schema_name=None (default), it defaults to *DEFAULT_SCHEMA* (i.e. 'public')
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

See also:

- Examples for the method *PostgreSQL.create_table()*.

PostgreSQL.get_column_dtype

PostgreSQL.**get_column_dtype**(*table_name*, *column_names=None*, *schema_name=None*)

Get information about data types of all or specific columns of a table.

Parameters

- **table_name** (*str*) – name of a table
- **column_names** (*str or list or None*) – (list of) column name(s); when column_names=None (default), all available columns are included
- **schema_name** – name of a schema; when 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 or None

See also:

- Examples for the method *PostgreSQL.create_table()*.

PostgreSQL.get_column_info

PostgreSQL.**get_column_info**(*table_name*, *schema_name=None*, *as_dict=True*)

Get information about columns of a table.

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str or None*) – name of a schema; when 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 or dict

See also:

- Examples for the method *PostgreSQL.create_table()*.

PostgreSQL.get_database_names

PostgreSQL.**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 or pandas.DataFrame

Examples:

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

>>> # Connect the default database 'postgres'
>>> # postgres = PostgreSQL('localhost', 5432, 'postgres', database_name='postgres')
>>> postgres = PostgreSQL()
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/postgres ... Successfully.
```

(continues on next page)

(continued from previous page)

```
>>> 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*)

Get the size of a database.

Parameters

database_name (*str or None*) – name of a database; if database_name=None (default), the function returns the size of the currently-connected database

Returns

size of the database

Return type

int

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.
>>> testdb.get_database_size()
'8553 kB'

>>> testdb.DEFAULT_DATABASE
'postgres'
>>> testdb.get_database_size(database_name=testdb.DEFAULT_DATABASE)
'8577 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*)

Get the primary keys of a table.

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str or None*) – name of a 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 or pandas.DataFrame

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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 Fig. 14 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

Fig. 14: 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)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.get_schema_names

PostgreSQL.**get_schema_names**(*include_all=False*, *names_only=True*, *verbose=False*)

Get the names of existing schemas.

Parameters

- **include_all** (*bool*) – whether to list all the available schemas, defaults to False
- **names_only** (*bool*) – whether to return only the names of the schema names, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

schema names

Return type

list or pandas.DataFrame or None

Examples:

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

(continues on next page)

(continued from previous page)

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

>>> testdb.get_schema_names()
['public']

>>> testdb.get_schema_names(include_all=True, names_only=False)
          schema_name  schema_id      role
0  information_schema      13388  postgres
1          pg_catalog         11  postgres
2            pg_toast         99  postgres
3              public       2200  postgres

>>> testdb.drop_schema(schema_names='public', verbose=True)
To drop the schema "public" from postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Dropping "public" ... Done.

>>> testdb.get_schema_names()  # None

>>> testdb.get_schema_names(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*)

Get the names of all tables in a schema.

Parameters

- **schema_name** (*str or None*) – name of a schema; when schema_name=None (default), it defaults to *DEFAULT_SCHEMA* (i.e. 'public')
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns

a list of table names

Return type

list or None

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> lst_tbl_names = testdb.get_table_names()
>>> lst_tbl_names
[]

>>> lst_tbl_names = testdb.get_table_names(schema_name='testdb', verbose=True)
The schema "testdb" does not exist.

>>> # 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.

>>> lst_tbl_names = testdb.get_table_names(schema_name='public')
>>> lst_tbl_names
['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.import_data

PostgreSQL.**import_data**(*data*, *table_name*, *schema_name=None*, *if_exists='fail'*, *force_replace=False*, *chunk_size=None*, *col_type=None*, *method='multi'*, *index=False*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Import tabular data into a table.

See also [DBMS-PS-ID-1] and [DBMS-PS-ID-2].

Parameters

- **data** (*pandas.DataFrame or pandas.io.parsers.TextFileReader or list or tuple*) – tabular data to be dumped into a database
- **table_name** (*str*) – name of a table
- **schema_name** (*str*) – name of a schema; when `schema_name=None` (default), it defaults to *DEFAULT_SCHEMA* (i.e. `'public'`)
- **if_exists** (*str*) – if the table already exists, to `'replace'`, `'append'` or, by default, `'fail'` and do nothing but raise a ValueError.
- **force_replace** (*bool*) – whether to force replacing existing table, defaults to `False`

- **chunk_size** (*int or None*) – the number of rows in each batch to be written at a time, defaults to None
- **col_type** (*dict or None*) – data types for columns, defaults to None
- **method** (*str or None or Callable*) – method for SQL insertion clause, defaults to 'multi'
  - None: uses standard SQL INSERT clause (one per row);
  - 'multi': pass 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 dump the index as a column
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False
- **kwargs** – [optional] parameters of pandas.DataFrame.to_sql

See also:

- Examples for the method *PostgreSQL.read_sql_query()*.

PostgreSQL.null_text_to_empty_string

PostgreSQL.**null_text_to_empty_string**(*table_name*, *column_names=None*, *schema_name=None*)

Convert null values (in text columns) to empty strings.

Parameters

- **table_name** (*str*) – name of a table
- **column_names** (*str or list or None*) – (list of) column name(s); when column_names=None (default), all available columns are included
- **schema_name** (*str*) – name of a schema

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
Password (postgres@localhost:5432): ***
Creating a database: "testdb" ... Done.
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> dat = example_dataframe()
>>> dat.Longitude = dat.Longitude.astype(str)
```

(continues on next page)

(continued from previous page)

```
>>> 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'}
```

	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

Fig. 15: 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)
```

	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

Fig. 16: 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
```

(continues on next page)

(continued from previous page)

```
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgreSQL.psql_insert_copy

static PostgreSQL.**psql_insert_copy**(*sql_table*, *sql_db_engine*, *column_names*, *data_iter*)

A callable using PostgreSQL COPY clause for executing inserting data.

Parameters

- **sql_table** – pandas.io.sql.SQLTable
- **sql_db_engine** – sqlalchemy.engine.Connection or sqlalchemy.engine.Engine
- **column_names** – (list of str) column names
- **data_iter** – iterable that iterates the values to be inserted

---

Note: This function is copied and slightly 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*)

Read table data by SQL query (recommended for large table).

See also [DBMS-PS-RSQ-1], [DBMS-PS-RSQ-2] and [DBMS-PS-RSQ-3].

Parameters

- **sql_query** (*str*) – a SQL query to be executed
- **method** (*str*) – method to be used for buffering temporary data
  - `'tempfile'` (default): use tempfile.TemporaryFile
  - `'stringio'`: use io.StringIO
  - `'spooled'`: use tempfile.SpooledTemporaryFile
- **max_size_spooled** (*int or float*) – `max_size` of tempfile.SpooledTemporaryFile, defaults to `1` (in gigabyte)
- **delimiter** (*str*) – delimiter used in data, defaults to `','`
- **tempfile_kwargs** (*dict or None*) – [optional] parameters of tempfile.TemporaryFile or tempfile.SpooledTemporaryFile

- **stringio_kwargs** (*dict or None*) – [optional] parameters of io.StringIO, e.g. `initial_value` (default: `''`)
- **kwargs** – [optional] parameters of pandas.read_csv

Returns
data frame as queried by the statement `sql_query`

Return type
pandas.DataFrame

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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
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)
To import data into "points"."England" at postgres:***@localhost:5432/testdb
? [No]|Yes: yes
Creating a schema: "points" ... Done.
Importing the data into the table "points"."England" ... Done.
```

The table *"points"."England"* is illustrated in Fig. 14 below:

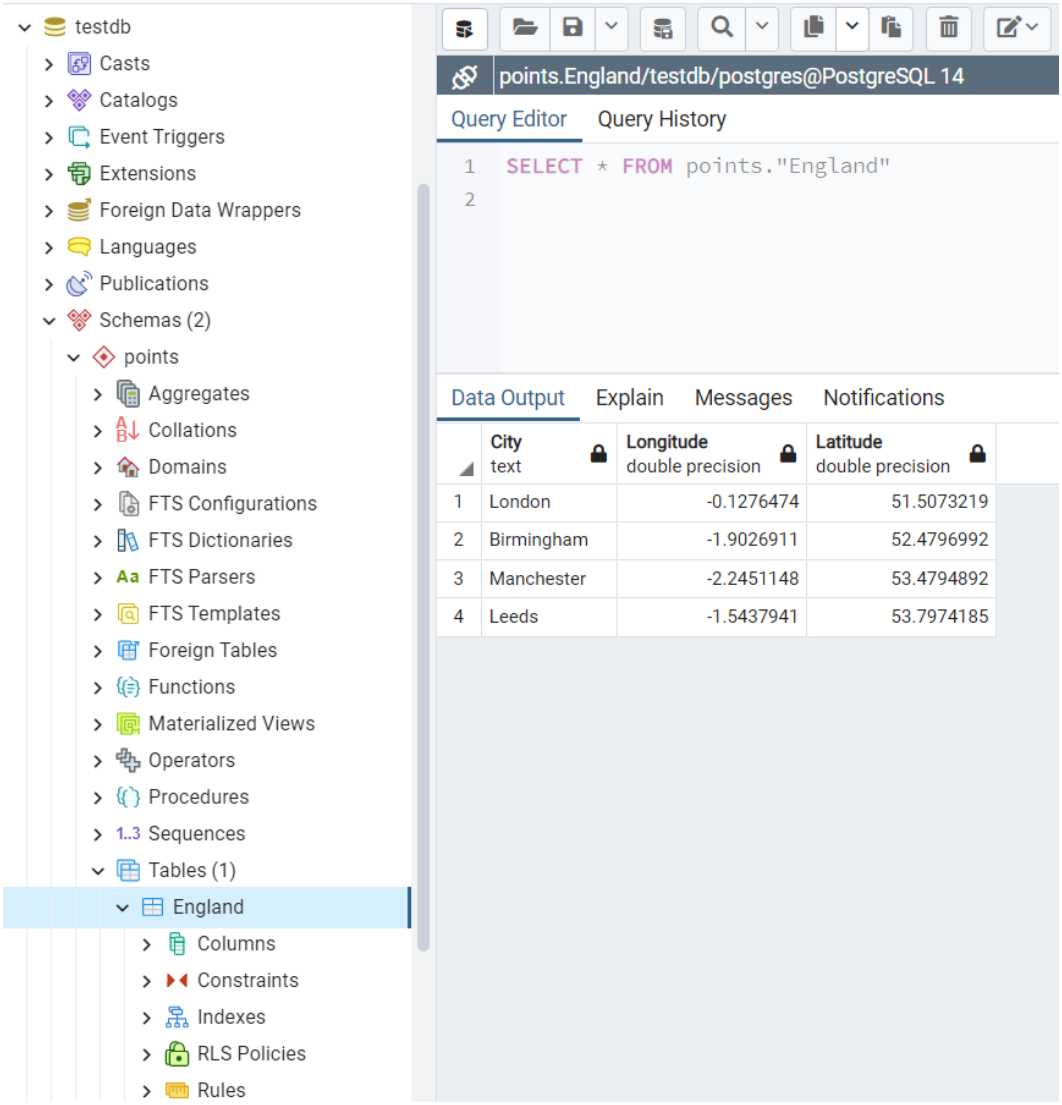

Fig. 17: The table "*points*"."*England*" in the database "*testdb*".

```
>>> rslt = testdb.table_exists(table_name=table, schema_name=schema)
>>> print(f"The table "{schema}"."{table}" exists? {rslt}.")
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
```

(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

>>> 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*)

Read data from a table.

See also [DBMS-PS-RT-1].

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str*) – name of a schema; when `schema_name=None` (default), it defaults to *DEFAULT_SCHEMA* (i.e. `'public'`)

- **conditions** (*str or None*) – defaults to None
- **chunk_size** (*int or None*) – number of rows to include in each chunk, defaults to None
- **sorted_by** (*str or None*) – name(s) of a column (or columns) by which the retrieved data is sorted, defaults to None
- **kwargs** – [optional] parameters of the method *read_sql_query()* or the function pandas.read_sql()

Returns
data frame from the specified table

Return type
pandas.DataFrame

See also:

- Examples for the method *PostgreSQL.read_sql_query()*.

PostgreSQL.schema_exists

PostgreSQL.**schema_exists**(*schema_name*)

Check whether a schema exists.

Parameters
schema_name (*str*) – name of a schema

Returns
whether the schema exists

Return type
bool

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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*)

Check whether a table exists.

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str*) – name of a schema; when schema_name=None (default), it defaults to *DEFAULT_SCHEMA* (i.e. 'public')

Returns

whether the table exists in the currently-connected database

Return type

bool

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, 'postgres', database_name='testdb')
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.
```

MSSQL

class pyhelpers.dbms.**MSSQL**(*host=None*, *port=None*, *username=None*, *password=None*, *database_name=None*, *confirm_db_creation=False*, *verbose=True*)

A class for basic communication with Microsoft SQL Server databases.

Parameters

- **host** (*str or None*) – name of the server running the SQL Server, e.g. `'localhost'` or `'127.0.0.1'`; when `host=None` (default), it is initialized as `'localhost'`
- **port** (*int or None*) – listening port; when `port=None` (default), it is initialized as `1433` (default by installation of the SQL Server)
- **username** (*str or None*) – name of the user or login used to connect; when `username=None` (default), the instantiation relies on Windows Authentication
- **password** (*str or int or None*) – user's password; when `password=None` (default), it is required to mannually type in the correct password to connect the PostgreSQL server
- **database_name** (*str or None*) – name of a database; when `database=None` (default), it is initialized as `'master'`
- **confirm_db_creation** (*bool*) – whether to prompt a confirmation before creating a new database (if the specified database does not exist), defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `True`

Variables

- **host** (*str*) – host name/address
- **port** (*str*) – listening port used by PostgreSQL
- **username** (*str*) – username
- **database_name** (*str*) – name of a database
- **credentials** (*dict*) – basic information about the server/database being connected
- **auth** (*str or None*) – authentication method (used for establish the connection)
- **address** (*str*) – representation of the database address
- **engine** (*sqlalchemy.engine.Engine*) – a SQLAlchemy connectable engine to a SQL Server; see also [DBMS-MS-3]

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
Connecting <server_name>@localhost:1433/master ... Successfully.

>>> mssql.address
'<server_name>@localhost:1433/master'

>>> 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)
To drop the database [testdb] from <server_name>@localhost:1433
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

Attributes

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.DEFAULT_DATABASE

MSSQL.**DEFAULT_DATABASE = 'master'**

Default database name.

MSSQL.DEFAULT_DIALECT

MSSQL.**DEFAULT_DIALECT = '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 = 'pyodbc'**

Default name of database driver. See also [DBMS-MS-2].

MSSQL.DEFAULT_HOST

MSSQL.**DEFAULT_HOST = 'localhost'**

Default host (server name). Alternatively, `os.environ['COMPUTERNAME']`

MSSQL.DEFAULT_ODBC_DRIVER

MSSQL.**DEFAULT_ODBC_DRIVER = 'ODBC Driver 17 for SQL Server'**

Default ODBC driver.

MSSQL.DEFAULT_PORT

MSSQL.**DEFAULT_PORT = 1433**

Default listening port used by Microsoft SQL Server.

MSSQL.DEFAULT_SCHEMA

MSSQL.**DEFAULT_SCHEMA = 'dbo'**

Default schema name.

MSSQL.DEFAULT_USERNAME

MSSQL.**DEFAULT_USERNAME = 'sa'**

Default username.

Methods

connect_database([database_name, verbose])	Establish a connection to a database.
create_connection([database_name, ...])	Create a SQLAlchemy connection.
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.
create_schema(schema_name[, verbose])	Create a schema.
create_table(table_name, column_specs[, ...])	Create a table.
database_exists([database_name])	Check whether a database exists.
disconnect_database([database_name, verbose])	Disconnect a database.
drop_database([database_name, ...])	Delete/drop a database.
drop_table(table_name[, schema_name, ...])	Delete/drop a table.
get_column_names(table_name[, schema_name])	Get column names of a table.
get_database_names([names_only])	Get names of all existing databases.
get_file_tables([names_only])	Get information about *FileTables* (if available).
get_primary_keys([table_name, schema_name, ...])	Get the primary keys of table(s).
get_row_count(table_name[, schema_name])	Get row count of a table in a database.
get_table_names([schema_name])	Get names of all tables stored in a schema.
has_dtypes(table_name, dtypes[, schema_name])	Check whether a table contains data of a certain data type or data types.
import_data(data, table_name[, schema_name, ...])	Import tabular data into a table.
read_columns(table_name, column_names[, ...])	Read data of specific columns of a table.
read_table(table_name[, schema_name, ...])	Read data from a table.
schema_exists(schema_name)	Check whether a schema exists.
specify_conn_str([database_name, auth, password])	Specify a string used for establishing a connection.
table_exists(table_name[, schema_name])	Check whether a table exists.

MSSQL.connect_database

MSSQL.**connect_database**(*database_name=None*, *verbose=False*)

Establish a connection to a database.

Parameters

- **database_name** (*str or None*) – name of a database; when database_name=None (default), the database name is input manually
- **verbose** (*bool or int*) – whether to print relevant information in 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.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*, *close_with_result=False*, *mode=None*)

Create a SQLAlchemy connection.

Parameters

- **database_name** (*str or None*) – name of a database, defaults to the name of the currently-connected database when database=None
- **close_with_result** (*bool*) – parameter of the method sqlalchemy.engine.Engine.connect(), defaults to False

- **mode** (*None or str*) – when `mode=None` (default), the method uses the existing engine; when `mode='pyodbc'` (optional), it uses pyodbc.connect()

Returns
a SQLAlchemy connection to a Microsoft SQL Server

Return type
sqlalchemy.engine.Connection or pyodbc.Connection

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
Connecting <server_name>@localhost:1433/master ... Successfully.

>>> db_conn = mssql.create_connection()
>>> db_conn.should_close_with_result
False
>>> db_conn.closed
False
>>> rslt = db_conn.execute('SELECT 1')
>>> rslt.fetchall()
[(1,)]
>>> db_conn.closed
False
>>> db_conn.close()
>>> db_conn.closed
True

>>> db_conn = mssql.create_connection()
>>> db_conn.should_close_with_result
True
>>> db_conn.closed
False
>>> rslt = db_conn.execute('SELECT 1')
>>> rslt.fetchall()
[(1,)]
>>> db_conn.closed
True
```

MSSQL.create_cursor

MSSQL.**create_cursor**(*database_name=None*)

Create a pyodbc cursor.

Parameters
database_name (*str or None*) – name of a database, defaults to the name of the currently-connected database when `database=None`

Returns
a pyodbc cursor

Return type
pyodbc.Cursor

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
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 a database
- **verbose** (*bool or int*) – whether to print relevant information in 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'
```

(continues on next page)

(continued from previous page)

```
>>> # 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.

Connect string format: '*mssql+pyodbc://<username>:<password>@<dsn_name>*'

Parameters

- **database_name** (*str or None*) – name of a database, defaults to the name of the currently-connected database when `database=None`
- **auth** (*str or None*) – authentication method (used for establish the connection), defaults to the current authentication method when `auth=None`
- **password** (*str or int or None*) – user's password; when `password=None` (default), it is required to manually type in the correct password to connect the PostgreSQL server

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()
Connecting <server_name>@localhost:1433/master ... Successfully.
```

(continues on next page)

(continued from previous page)

```
>>> db_engine = mssql.create_engine()
>>> db_engine.name
'mssql'

>>> db_engine.dispose()
```

MSSQL.create_schema

MSSQL.**create_schema**(*schema_name*, *verbose=False*)

Create a schema.

Parameters

- **schema_name** (*str*) – name of a schema in the currently-connected database
- **verbose** (*bool or int*) – whether to print relevant information in 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.

>>> 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
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

MSSQL.create_table

MSSQL.**create_table**(*table_name*, *column_specs*, *schema_name=None*, *verbose=False*)

Create a table.

Parameters

- **table_name** (*str*) – name of a table
- **column_specs** (*str*) – specifications for each column of the table
- **schema_name** (*str or None*) – name of a schema; when schema_name=None (default), it defaults to *DEFAULT_SCHEMA* (i.e. 'dbo')

- **verbose** (*bool or int*) – whether to print relevant information in 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.

>>> 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']

>>> # 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
? [No]|Yes: yes
Dropping [testdb] ... Done.
```

MSSQL.database_exists

MSSQL.**database_exists**(*database_name=None*)

Check whether a database exists.

Parameters

database_name (*str or None*) – name of a database, 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.
```

(continues on next page)

(continued from previous page)

```
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*)

Disconnect a database.

Parameters

- **database_name** (*str or None*) – name of database to disconnect from; if database_name=None (default), disconnect the current database.
- **verbose** (*bool or int*) – whether to print relevant information in 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.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.
```

(continues on next page)

(continued from previous page)

```
>>> 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*)

Delete/drop a database.

Parameters

- **database_name** (*str or None*) – database to be disconnected; if database_name=None (default), drop the database being currently currented
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in 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.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_table

MSSQL.**drop_table**(*table_name*, *schema_name=None*, *confirmation_required=True*, *verbose=False*)

Delete/drop a table.

Parameters

- **table_name** (*str*) – name of a table

- **schema_name** (*str or None*) – name of a schema; when `schema_name=None` (default), it defaults to *DEFAULT_SCHEMA* (i.e. `'dbo'`)
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

See also:

- Examples for the method *MSSQL.create_table()*.

MSSQL.get_column_names

MSSQL.**get_column_names**(*table_name*, *schema_name=None*)

Get column names of a table.

Parameters

- **table_name** (*str*) – name of a table in the currently-connected database
- **schema_name** (*str or None*) – defaults to `None`

Returns

a list of column names

Return type

list

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
Connecting <server_name>@localhost:1433/master ... Successfully.

>>> mssql.get_table_names()
['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 or pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
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*)

Get information about *FileTables* (if available).

Parameters
names_only (*bool*) – whether to return *FileTables* names only, defaults to `True`

Returns
information about *FileTables* (if available)

Return type
list or pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
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'*)

Get the primary keys of table(s).

Parameters

- **table_name** (*str or None*) – name of a table in the currently-connected database; when `table_name=None` (default), get the primary keys of all existing tables
- **schema_name** (*str or None*) – name of a schema, defaults to *DEFAULT_SCHEMA* when `schema_name=None`
- **table_type** (*str*) – table type, defaults to `'TABLE'`

Returns

a list of primary keys

Return type

list

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
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 row count of a table in a database.

Parameters

- **table_name** (*str*) – name of a table in the currently-connected database
- **schema_name** (*str or None*) – schema name of the given table, defaults to `None`

Returns

count of rows in the given table

Return type

int

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
Connecting <server_name>@localhost:1433/master ... Successfully.

>>> mssql.get_table_names()
['MSreplication_options',
 'spt_fallback_db',
 'spt_fallback_dev',
 'spt_fallback_usg',
 'spt_monitor']

>>> mssql.get_row_count(table_name='MSreplication_options')
3
```

MSSQL.get_table_names

MSSQL.**get_table_names**(*schema_name=None*)

Get names of all tables stored in a schema.

Parameters

schema_name (*str or list or None*) – name of a schema, defaults to *DEFAULT_SCHEMA* when schema_name=None

Returns

names of tables in the given schema mssql_schema_name

Return type

list

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
Connecting <server_name>@localhost:1433/master ... Successfully.

>>> mssql.get_table_names()
['MSreplication_options',
 'spt_fallback_db',
 'spt_fallback_dev',
 'spt_fallback_usg',
 'spt_monitor']

>>> mssql.get_table_names(schema_name=['dbo', 'sys'])
```

MSSQL.has_dtypes

MSSQL.**has_dtypes**(*table_name*, *dtypes*, *schema_name=None*)

Check whether a table contains data of a certain data type or data types.

Parameters

- **table_name** (*str*) – name of a table in the currently-connected database
- **dtypes** (*str or Sequence[str]*) – data types, such as `'geometry'`, `'hierarchyid'`, `'varbinary'`
- **schema_name** (*str or None*) – name of a schema, defaults to *DEFAULT_SCHEMA* when `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()
Connecting <server_name>@localhost:1433/master ... Successfully.

>>> mssql.get_table_names()
['MSreplication_options',
 'spt_fallback_db',
 'spt_fallback_dev',
 'spt_fallback_usg',
 'spt_monitor']

>>> rslt = mssql.has_dtypes(table_name='spt_monitor', dtypes='varbinary')
>>> list(rslt)
[('varbinary', False, [])]

>>> rslt = mssql.has_dtypes(table_name='spt_monitor', dtypes=['geometry', 'int'])
>>> list(rslt)
[('geometry', False, []),
 ('int',
  True,
  ['cpu_busy',
   'io_busy',
   'idle',
   'pack_received',
   'pack_sent',
   '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*, *col_type=None*, *method='multi'*, *index=False*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Import tabular data into a table.

See also [DBMS-MS-ID-1].

Parameters

- **data** (*pandas.DataFrame or pandas.io.parsers.TextFileReader or list or tuple*) – tabular data to be dumped into a database
- **table_name** (*str*) – name of a table
- **schema_name** (*str or None*) – name of a schema; when `schema_name=None` (default), it defaults to *DEFAULT_SCHEMA* (i.e. `'public'`)
- **if_exists** (*str*) – if the table already exists, to `'replace'`, `'append'` or, by default, `'fail'` and do nothing but raise a ValueError.
- **force_replace** (*bool*) – whether to force replacing existing table, defaults to `False`
- **chunk_size** (*int or None*) – the number of rows in each batch to be written at a time, defaults to `None`
- **col_type** (*dict or None*) – data types for columns, defaults to `None`
- **method** (*str or None or Callable*) – method for SQL insertion clause, defaults to `'multi'`
  - `None`: uses standard SQL `INSERT` clause (one per row);
  - `'multi'`: pass 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 dump the index as a column
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – [optional] parameters of pandas.DataFrame.to_sql

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> from pyhelpers._cache import example_dataframe

>>> testdb = MSSQL(database_name='testdb')
```

(continues on next page)

(continued from previous page)

```
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, 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.
```

The imported example data can also be viewed using Microsoft SQL Server Management Studio (as illustrated in Fig. 18 below):

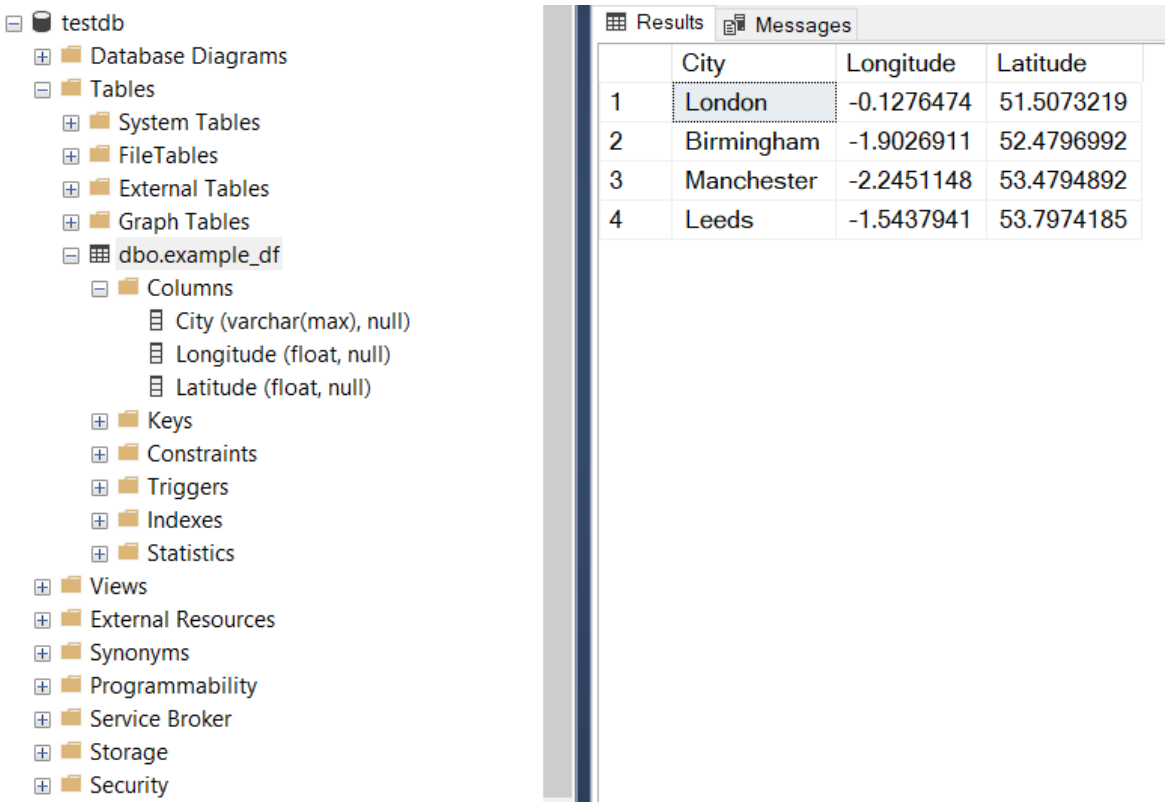

Fig. 18: The table *[dbo].[example_df]* in the database *[testdb]*.

```
>>> # 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
```

(continues on next page)

(continued from previous page)

```
? [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 *MSSQL.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 or tuple*) – column name(s) of the specified table
- **dtype** (*str or None*) – data type, defaults to None; options include 'hierarchyid', 'varbinary' and 'geometry'
- **schema_name** (*str or None*) – name of a schema, defaults to *DEFAULT_SCHEMA* when schema_name=None
- **chunk_size** (*int or None*) – number of rows to include in each chunk (if specified), defaults to None
- **kwargs** – [optional] parameters of 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()
Connecting <server_name>@localhost:1433/master ... Successfully.

>>> mssql.get_table_names()
['MSreplication_options',
 'spt_fallback_db',
```

(continues on next page)

(continued from previous page)

```
 '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 *MSSQL.read_table()*.

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*, ***kwargs*)

Read data from a table.

Parameters

- **table_name** (*str*) – name of a table in the currently-connected database
- **schema_name** (*str or None*) – name of a schema, defaults to *DEFAULT_SCHEMA* when schema_name=None
- **column_names** (*list or tuple or None*) – column name(s), defaults to all columns when column_names=None
- **conditions** (*str or None*) – conditions in a SQL query statement, defaults to None
- **chunk_size** (*int or None*) – number of rows to include in each chunk (if specified), defaults to None
- **save_as** (*str or None*) – file extension (if specified) for saving table data locally, defaults to None
- **data_dir** (*str or None*) – directory where the table data is to be saved, defaults to None
- **kwargs** – [optional] parameters of pandas.read_sql

Returns

data of the queried table

Return type

pandas.DataFrame

Examples:

```
>>> from pyhelpers.dbms import MSSQL
>>> from pyhelpers._cache import example_dataframe

>>> mssql = MSSQL()
Connecting <server_name>@localhost:1433/master ... Successfully.

>>> mssql.get_table_names()
['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, 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.

>>> # Retrieve the imported data
>>> example_df_ret = testdb.read_table(table_name=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]
```

(continues on next page)

(continued from previous page)

```
>>> 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 *MSSQL.import_data()*.

MSSQL.schema_exists

MSSQL.**schema_exists**(*schema_name*)

Check whether a schema exists.

Parameters
schema_name (*str*) – name of a schema in the currently-connected database

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 a string used for establishing a connection.

Parameters

- **database_name** (*str or None*) – name of a database, defaults to the name of the currently-connected database when `database=None`

- **auth** (*str or None*) – authentication method (used for establish the connection), defaults to the current authentication method when `auth=None`
- **password** (*str or int or None*) – user's password; when `password=None` (default), it is required to mannually type in the correct password to connect the PostgreSQL server

Returns
connection string

Return type
str

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
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};Trusted_...
```

MSSQL.table_exists

MSSQL.**table_exists**(*table_name*, *schema_name=None*)

Check whether a table exists.

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str or None*) – name of a schema, defaults to *DEFAULT_SCHEMA* when `schema_name=None`

Returns
whether the table exists in the currently-connected database

Return type
bool

Examples:

```
>>> from pyhelpers.dbms import MSSQL

>>> mssql = MSSQL()
Connecting <server_name>@localhost:1433/master ... Successfully.

>>> mssql.table_exists(table_name='test_table')
False

>>> mssql.get_table_names()
['MSreplication_options',
 'spt_fallback_db',
```

(continues on next page)

(continued from previous page)

```
 'spt_fallback_dev',
 'spt_fallback_usg',
 'spt_monitor']

>>> mssql.table_exists(table_name='MSreplication_options')
True
```

3.7.2 Database tools/utilities

make_database_address(host, port, username)	Make a string of a database address.
get_default_database_address(db_cls)	Get default database address of a given class in the current module.
mssql_to_postgresql(mssql, postgres[, ...])	Copy tables of a database from a Microsoft SQL server to a PostgreSQL server.

make_database_address

pyhelpers.dbms.**make_database_address**(*host*, *port*, *username*, *database_name=''*)

Make a string of a database address.

Parameters

- **host** (*str*) – host name/address of a PostgreSQL server
- **port** (*int or str*) – listening port used by PostgreSQL
- **username** (*str*) – username of a PostgreSQL server
- **database_name** (*str or None*) – name of a database, defaults to ""

Returns

address of a database

Return type

str

Examples:

```
>>> from pyhelpers.dbms import make_database_address

>>> db_addr = make_database_address('localhost', 5432, 'postgres', 'postgres')
>>> db_addr
'postgres:***@localhost:5432/postgres'
```

get_default_database_address

pyhelpers.dbms.**get_default_database_address**(*db_cls*)

Get default database address of a given class in the current module.

Parameters
db_cls (*object*) – a class representation of a database

Returns
default address of database

Return type
str

Examples:

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

>>> db_addr = get_default_database_address(db_cls=PostgreSQL)
>>> db_addr
'None:***@None:None'
```

mssql_to_postgresql

pyhelpers.dbms.**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*)

Copy tables of a database from a Microsoft SQL server to a PostgreSQL server.

Parameters

- **mssql** (pyhelpers.dbms.MSSQL) – name of a Microsoft SQL (source) database
- **postgres** (pyhelpers.dbms.PostgreSQL) – name of a PostgreSQL (destination) database
- **mssql_schema** (*str or None*) – name of a schema to be migrated from the SQl Server
- **postgres_schema** (*str or None*) – name of a schema to store the migrated data in the PostgreSQL server
- **chunk_size** (*int or None*) – number of rows in each batch to be read/written at a time, defaults to None
- **excluded_tables** (*list or None*) – names of tables that are excluded from the data migration
- **file_tables** (*bool*) – whether to include FileTables, defaults to False
- **memory_threshold** (*float or int*) – threshold (in GiB) beyond which the data is migrated by partitions, defaults to 2.

- **update** (*bool*) – whether to redo the transfer between the database servers, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information, defaults to `True`

Examples:

```
>>> from pyhelpers.dbms import mssql_to_postgresql, PostgreSQL, MSSQL
>>> from pyhelpers._cache import example_dataframe

>>> # Connect/create a PostgreSQL database, which is named [testdb]
>>> mssql_testdb = MSSQL(database_name='testdb')
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']
```

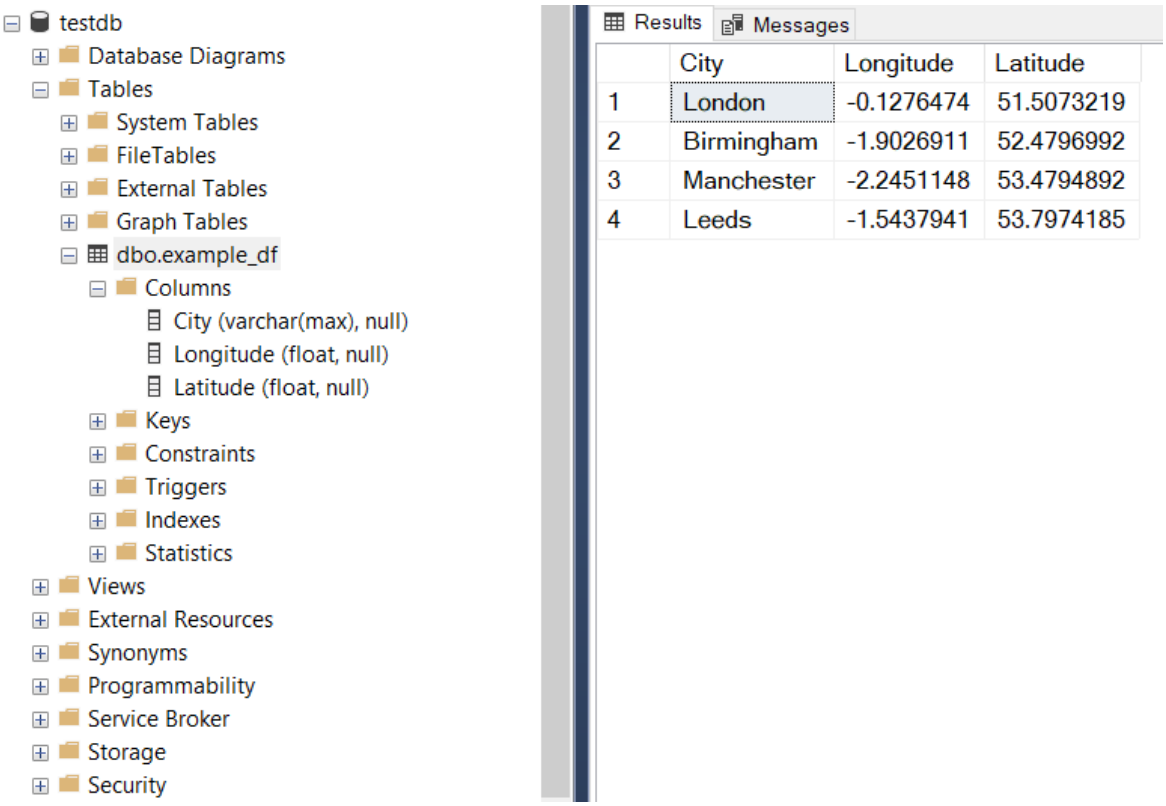

Fig. 19: The table *[dbo].[example_df]* in the Microsoft SQL Server database *[testdb]*.

```
>>> # Create an instance for a PostgreSQL database, which is also named "testdb"
>>> postgres_testdb = PostgreSQL(database_name='testdb')
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()
[]

>>> # 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()
['example_df']
```

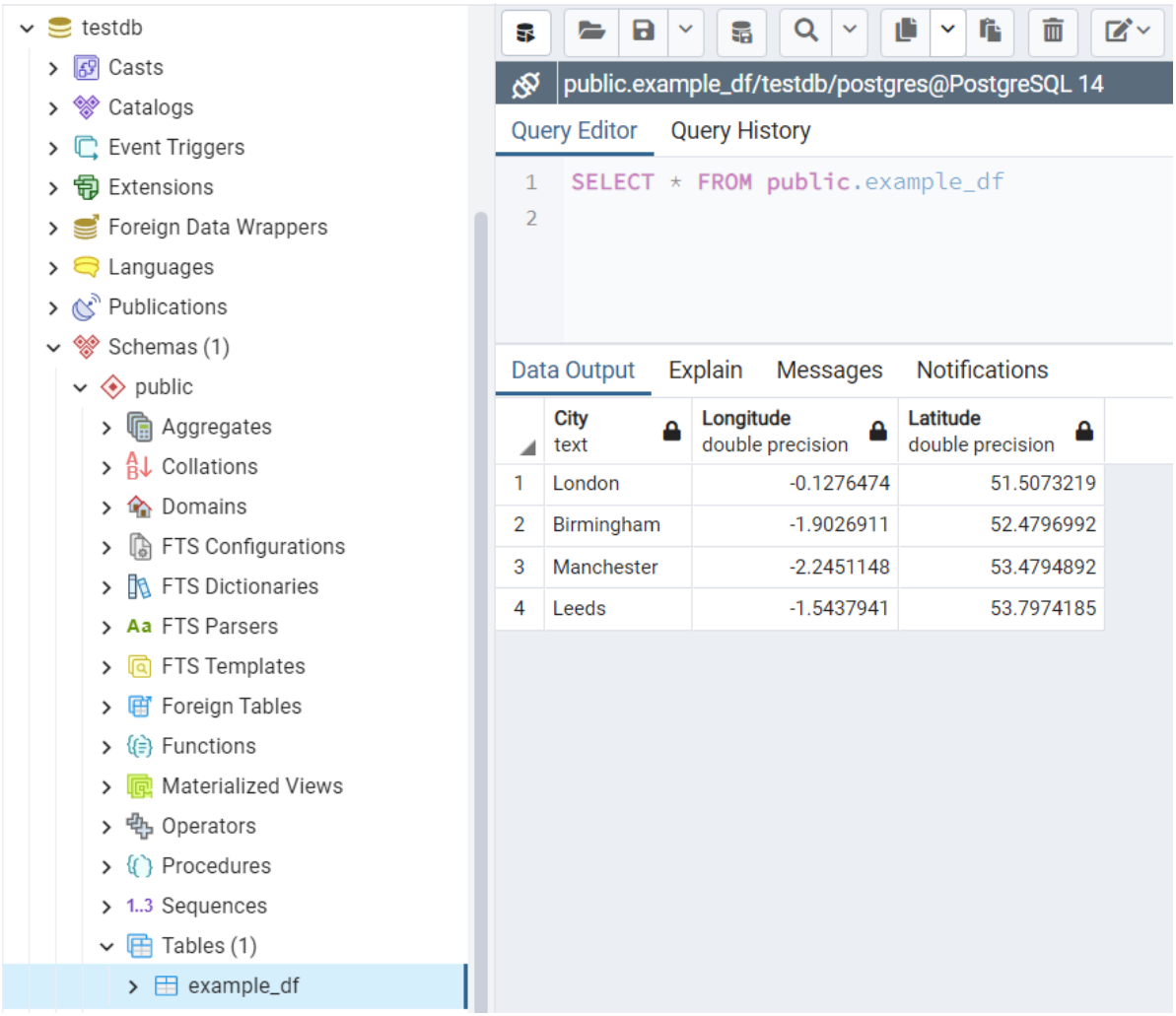

Fig. 20: The table *"dbo"."example_df"* in the PostgreSQL database *"testdb"*.

```
>>> # 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.
```

Chapter 4

License

PyHelpers is licensed under GNU General Public License v3 or later (GPLv3+).

Chapter 5

Tutorial

This brief tutorial provides a few examples for each of the *modules* to demonstrate that what pyhelpers could offer to assist us in performing data manipulation tasks in our day-to-day work.

5.1 Preparation - Create a data set

To begin with, let's use NumPy and Pandas to create an example data set, which is used throughout this tutorial.

Note:

- NumPy and Pandas are installed along with the installation of pyhelpers as they are among the dependencies of the package.

For demonstration purposes, firstly, we can use the function numpy.random.rand to generate a 100-by-100 Numpy array of random samples drawn from a standard uniform distribution, and name it `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(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,
        0.1419334 ],
       [0.88498232, 0.19701397, 0.56861333, ..., 0.75842952, 0.02378743,
```

(continues on next page)

(continued from previous page)

```
       0.81357508]])

>>> random_array.shape  # Check the shape of the array
(100, 100)
```

Then, we use the class pandas.DataFrame to transform `random_array` into a Pandas data frame, which is presented in tabular form, and name 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(100)])
>>> data_frame
      col_0     col_1     col_2  ...    col_97    col_98    col_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]
```

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

5.2 Alter settings for display of data

The module *pyhelpers.settings* can be used to alter a few frequently-used parameters (of GDAL, Matplotlib, NumPy and Pandas) such that the working environment is adapted to suit our own preferences. For example, we could apply the function *np_preferences()* with its default parameters whereby we may have a 'neater' view of the `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, ..., 0.1832, 0.5865, 0.0201, 0.8289, 0.0047],
       [0.6778, 0.2700, 0.7352, 0.9622, 0.2488, ..., 0.4905, 0.2274, 0.2544, 0.0580, 0.4344],
       [0.3118, 0.6963, 0.3778, 0.1796, 0.0247, ..., 0.2243, 0.0978, 0.8622, 0.9729, 0.9608],
       [0.9066, 0.7740, 0.3331, 0.0811, 0.4072, ..., 0.9590, 0.3554, 0.3567, 0.0163, 0.1852],
       [0.4013, 0.9293, 0.0996, 0.9453, 0.8695, ..., 0.2717, 0.4554, 0.4017, 0.2484, 0.5059],
       ...,
       [0.0299, 0.9851, 0.0947, 0.4510, 0.8387, ..., 0.1239, 0.2947, 0.3699, 0.9100, 0.1429],
       [0.6169, 0.2029, 0.2888, 0.4451, 0.5472, ..., 0.4776, 0.8664, 0.2150, 0.1436, 0.9332],
       [0.8911, 0.2687, 0.8403, 0.7570, 0.9954, ..., 0.9835, 0.4088, 0.5737, 0.7373, 0.2252],
       [0.2697, 0.7388, 0.8071, 0.2006, 0.3087, ..., 0.5063, 0.2319, 0.9484, 0.8813, 0.1419],
       [0.8850, 0.1970, 0.5686, 0.9310, 0.5645, ..., 0.5504, 0.3972, 0.7584, 0.0238, 0.8136]])
```

To reset the display, we can set `reset=True` by which the altered parameters are reset to their default values:

```
>>> 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]])
```

Note:

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

For another example, the function *pd_preferences()* alters a few Pandas options and settings, such as representation and maximum number of columns when displaying a Pandas DataFrame. Applying the function with its default parameters should allow us to view all the 100 columns and the precision of numbers changes 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  col_8  col_9  col_10  col_11 ...
0  0.5488 0.7152 0.6028 0.5449 0.4237 0.6459 0.4376 0.8918 0.9637 0.3834  0.7917  0.5289 ...
1  0.6778 0.2700 0.7352 0.9622 0.2488 0.5762 0.5920 0.5723 0.2231 0.9527  0.4471  0.8464 ...
2  0.3118 0.6963 0.3778 0.1796 0.0247 0.0672 0.6794 0.4537 0.5366 0.8967  0.9903  0.2169 ...
3  0.9066 0.7740 0.3331 0.0811 0.4072 0.2322 0.1325 0.0534 0.7256 0.0114  0.7706  0.1469 ...
4  0.4013 0.9293 0.0996 0.9453 0.8695 0.4542 0.3267 0.2327 0.6145 0.0331  0.0156  0.4288 ...
..    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...     ...     ... ...
95 0.0299 0.9851 0.0947 0.4510 0.8387 0.4216 0.2488 0.4140 0.8239 0.0449  0.4888  0.1935 ...
96 0.6169 0.2029 0.2888 0.4451 0.5472 0.1754 0.5955 0.6072 0.4085 0.2007  0.3339  0.0980 ...
97 0.8911 0.2687 0.8403 0.7570 0.9954 0.1634 0.8974 0.0570 0.6731 0.6692  0.9157  0.2279 ...
98 0.2697 0.7388 0.8071 0.2006 0.3087 0.0087 0.3848 0.9011 0.4013 0.7590  0.0574  0.5879 ...
99 0.8850 0.1970 0.5686 0.9310 0.5645 0.2116 0.2650 0.6786 0.7470 0.5918  0.2814  0.1868 ...

[100 rows x 100 columns]
```

Note:

- Here the columns from `'col_12'` to `'col_99'` are omitted from the above demonstration due to the limit of the page width.

Similarly, the function *pd_preferences()* also offers a parameter `reset`, which defaults to `False`; by setting `reset=True`, the altered parameters are reset to their default values. In addition, we can also set `reset='all'` to reset all Pandas options to their default values, if needed.

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

>>> data_frame
       col_0     col_1     col_2  ...    col_97    col_98    col_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:

- The functions that are currently available in the module *pyhelpers.settings* handle only a few parameters for the author's personal preference. We may change the source code as appropriate to adapt the settings to different tastes.

5.3 Specify a directory or a file path

The module *pyhelpers.dirs* offers to assist with manipulating folders/directories. For example, the function *cd()* returns an absolute path to the current working directory or, if specified, to a subdirectory or a file any level deep from the current working directory:

```
>>> from pyhelpers.dirs import cd
>>> 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"
```

(continues on next page)

(continued from previous page)

```
>>> # 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'The directory "{rel_dir_path}\\" exists? {os.path.exists(path_to_dir)}')
The directory "pyhelpers_tutorial\" exists? False
```

If the directory "pyhelpers_tutorial\" does not exist, we could set the parameter mkdir=True by which the directory should be created as we specify the path:

```
>>> # Set `mkdir` to be `True` to create a folder named "pyhelpers_tutorial"
>>> path_to_dir = cd(dir_name, mkdir=True)

>>> # Check again whether the directory "pyhelpers_tutorial\" exists
>>> print(f'The directory "{rel_dir_path}\\" exists? {os.path.exists(path_to_dir)}')
The directory "pyhelpers_tutorial\" exists? True
```

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

```
>>> # Name of a 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)
>>> print(rel_file_path)
pyhelpers_tutorial\quick_start.dat
```

If any of the folders/subfolders of a specified path does not exist, setting mkdir=True should enable the function *cd()* to create all the missing ones along the path. For example, let's specify a data directory, named "data", which is contained within the folder "pyhelpers_tutorial":

```
>>> # Path to a data directory
>>> data_dir = cd("pyhelpers_tutorial", "data")  # equivalent to `cd(path_to_dir, "data")`

>>> # Relative path of the data directory
>>> rel_data_dir = os.path.relpath(data_dir)
>>> print(rel_data_dir)
pyhelpers_tutorial\data
```

We can use the function *is_dir()* to examine whether data_dir (or rel_data_dir) specifies a path (or a relative path) to a directory:

```
>>> from pyhelpers.dirs import is_dir

>>> # Check whether `rel_data_dir` specifies a (relative) path to a directory
>>> print(f'`rel_data_dir` specifies a directory pathname? {is_dir(rel_data_dir)}')
`rel_data_dir` specifies a directory pathname? True

>>> # Check whether the data directory exists
>>> print(f'The directory "{rel_data_dir}\\" exists? {os.path.exists(rel_data_dir)}')
The directory "pyhelpers_tutorial\data\" exists? False
```

For another example, let's specify a path to a Pickle file, named `"dat.pickle"`, in the directory `"pyhelpers_tutorial\data\"`:

```
>>> # Filename of a Pickle file
>>> pickle_filename = "dat.pickle"

>>> # Path to the Pickle file, i.e. cd("pyhelpers_tutorial", "data", "dat.pickle")
>>> path_to_pickle = cd(data_dir, pickle_filename)

>>> # Relative path of the Pickle file
>>> rel_pickle_path = os.path.relpath(path_to_pickle)
>>> print(rel_pickle_path)
pyhelpers_tutorial\data\dat.pickle
```

Examine `rel_pickle_path` (or `path_to_pickle`):

```
>>> # Check whether `rel_pickle_path` specifies a directory
>>> print(f'`rel_pickle_path` specifies a directory? {os.path.isdir(rel_pickle_path)}')
`rel_pickle_path` specifies a directory? False

>>> # Check whether the file "dat.pickle" exists
>>> print(f'The file "{rel_pickle_path}" exists? {os.path.exists(rel_pickle_path)}')
The file "pyhelpers_tutorial\data\dat.pickle" exists? False
```

Let's now set the parameter `mkdir` to be `True`:

```
>>> path_to_pickle = cd(data_dir, pickle_filename, mkdir=True)
>>> rel_data_dir = os.path.relpath(data_dir)

>>> # Check again whether the data directory exists
>>> print(f'The directory "{rel_data_dir}\\" exists? {os.path.exists(rel_data_dir)}')
The directory "pyhelpers_tutorial\data\" exists? True

>>> # Check again whether the file "dat.pickle" exists
>>> print(f'The file "{rel_pickle_path}" exists? {os.path.exists(rel_pickle_path)}')
The file "pyhelpers_tutorial\data\dat.pickle" exists? False
```

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

To delete the directory *"pyhelpers_tutorial"* (and all contained within it), we can use the function *delete_dir()*:

```
>>> from pyhelpers.dirs import delete_dir

>>> # Delete the directory "pyhelpers_tutorial\"
```

(continues on next page)

(continued from previous page)

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

5.4 Save data to / load data from a Pickle file

The module *pyhelpers.store* can facilitate tasks such as saving our data to, and loading the data from, file-like objects of some popular formats, such as CSV, JSON and Pickle. For example, we could save the `data_frame` that has been created in the [Preparation](#tutorial-preparation) section as a Pickle file by using the functions *save_pickle()* and retrieve it later by using *load_pickle()*. Firstly, let's save `data_frame` to `path_to_pickle`, which has been specified in the *Specify a directory or a file path* section:

```
>>> from pyhelpers.store import save_pickle, load_pickle

>>> # Write `data_frame` to the file "dat.pickle"
>>> save_pickle(data_frame, path_to_pickle, verbose=True)
Saving "dat.pickle" to "pyhelpers_tutorial\data\" ... Done.
```

Now, we can retrieve the data from `path_to_pickle` and store the retrieved data in another variable named `df_retrieved`:

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

Check whether `df_retrieved` is equal to `data_frame` (namely, whether they have the same shape and elements):

```
>>> print(f'`df_retrieved` is equal to `data_frame`? {df_retrieved.equals(data_frame)}')
`df_retrieved` is equal to `data_frame`? True
```

Before we move on, let's delete again the Pickle file (i.e. `path_to_pickle`) and the directory created in the above example:

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

5.5 Convert coordinates between OSGB36 and WGS84

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

```
>>> from pyhelpers.geom import osgb36_to_wgs84

>>> # To convert coordinate of a single point (530034, 180381):
>>> easting, northing = 530039.558844, 180371.680166  # London

>>> longitude, latitude = osgb36_to_wgs84(easting, northing)  # Longitude and latitude
>>> (longitude, latitude)
(-0.12764738750268856, 51.507321895400686)
```

We could also use the function to convert 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, we can convert from the (longitude, latitude) back to (easting, northing) by 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 errors, which

are mostly negligible.

Check whether `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'`xy_array_` is almost equal to `xy_array`? {eq_res}')
`xy_array_` is almost equal to `xy_array`? True
```

5.6 Find similar texts

The module *pyhelpers.text* 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 Python built-in module - to perform the task. Alternatively, we could also make use of an open-source package, FuzzyWuzzy, via setting the parameter `engine='fuzzywuzzy'`:

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

Note:

- The package FuzzyWuzzy is NOT an essential dependency for the installation of pyhelpers>=1.3.0. We need to install it (e.g. via `pip install`) to make the function run successfully with setting `engine='fuzzywuzzy'`.
- In previous versions of pyhelpers (i.e. pyhelpers<=1.2.14), optional values for the parameter `engine` include `'fuzzywuzzy'` and `'nltk'`. The latter has been replaced with `'difflib'` since

pyhelpers 1.2.15.

5.7 Download an image file

The module *pyhelpers.ops* provides a miscellany of helper functions that may assist with various operations. For example, we can use the function *download_file_from_url()* to download a file from a given URL.

Let's now try to download an image file of Python logo from its home 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("pyhelpers_tutorial", "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)
```

The parameter `verbose` is by default `False`. If we set `verbose=True`, the function would print out relevant information about the download as the file is being downloaded.

Note:

- When `verbose=True` (or `verbose=1`), the function requires an open-source package tqdm, which is NOT an essential dependency for installing pyhelpers>=1.2.15. We can just install the dependency package via `pip install` to make the function run successfully.

Assuming tqdm has been installed in our working environment, try:

```
>>> download_file_from_url(url, python_logo_file_path, if_exists='replace', verbose=True)
"pyhelpers_tutorial\images\python-logo.png": 81.6kB [00:00, 10.8MB/s]
```

Note:

- '...MB/s' shown at the end of the output above 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 download the image file again and replace the one that was just downloaded to the specified destination.

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

```
>>> from PIL import Image

>>> python_logo = Image.open(python_logo_file_path)
>>> python_logo.show()
```

Fig. 21: The Python Logo (for illustration in the brief tutorial).

Note:

- In Jupyter Notebook, we may 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 the download directory `"pyhelpers_tutorial\images\"`, again, we can use the function *`delete_dir()`*:

```
>>> 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 having to type `yes` to confirm the action. This is actually implemented through the function *`confirmed()`*, which is also from the module *`pyhelpers.ops`* 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:

- What we type to respond to the prompting message is case-insensitive. It doesn't have to be precisely `Yes` to make the function return `True`; something like `yes`, `Y` or `ye` can do the job as well. If we type `no` or `n`, it returns `False`.

- The function also provides a parameter `confirmation_required`, which defaults to `True`. If setting `confirmation_required=False`, a confirmation is not required, in which case this function will become ineffective as it just returns `True`.

5.8 Work with a PostgreSQL server

The module *`pyhelpers.dbms`* offers a convenient way of communicating with databases, such as PostgreSQL.

Note:

- The current release of pyhelpers is only able to deal with PostgreSQL as the module *`pyhelpers.dbms`* needs to be further developed to work with other types of databases.

The class *`PostgreSQL`*, for example, could assist us in executing some basic SQL statements on a PostgreSQL database server. To demonstrate it works, let's start with importing the class:

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

5.8.1 Connect to a database

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

Note:

- If we leave `host`, `port`, `username` and `database_name` unspecified, their default arguments (namely, `host='localhost'`, `port=5432`, `username='postgres'` and `database_name='postgres'`) are passed to instantiate the class, in which case we would 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 along with the class instantiation.
- If we prefer not to specify explicitly the parameter `password`, we could just leave it. In that case, we will be asked to type in the password manually when instantiating the class.

For example, let's create an instance named `postgres`, and we'd like to establish a connection with a database named "*pyhelpers_tutorial*", which is hosted at the default PostgreSQL server:

```
>>> postgres = PostgreSQL(database_name="pyhelpers_tutorial")
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 now in the Databases tree of the default server, as illustrated in Fig. 22:

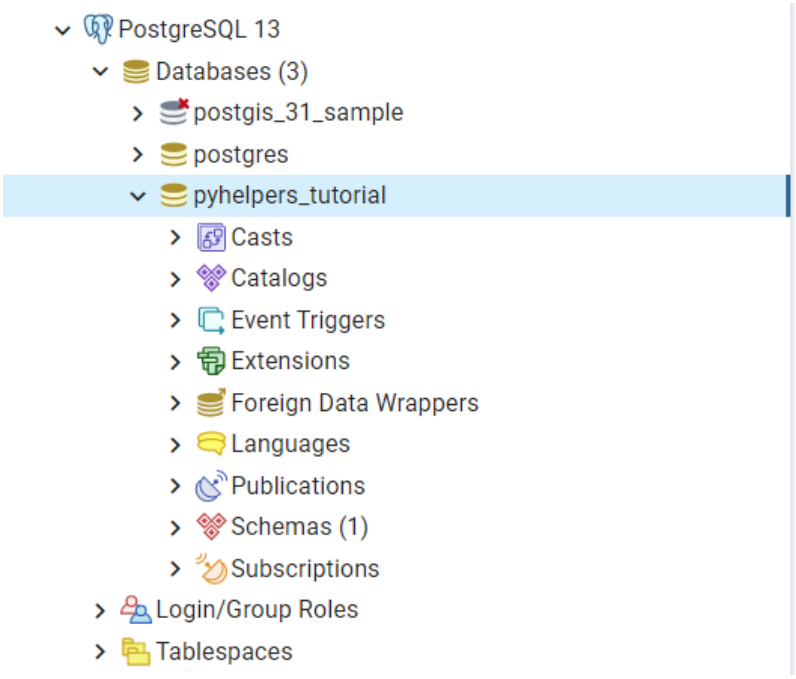

Fig. 22: The database "*pyhelpers_tutorial*".

Alternatively, we could also use the method `database_exists()`:

```
>>> res = postgres.database_exists("pyhelpers_tutorial")
>>> print(f'The database "pyhelpers_tutorial" exists? {res}')
The database "pyhelpers_tutorial" exists? True

>>> print(f'We are currently connected to the database "{postgres.database_name}".')
We are now connected with the database "pyhelpers_tutorial".
```

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

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

As we can see in Fig. 23, the database "*pyhelpers_tutorial_alt*" has now been added to the default *Databases* tree:

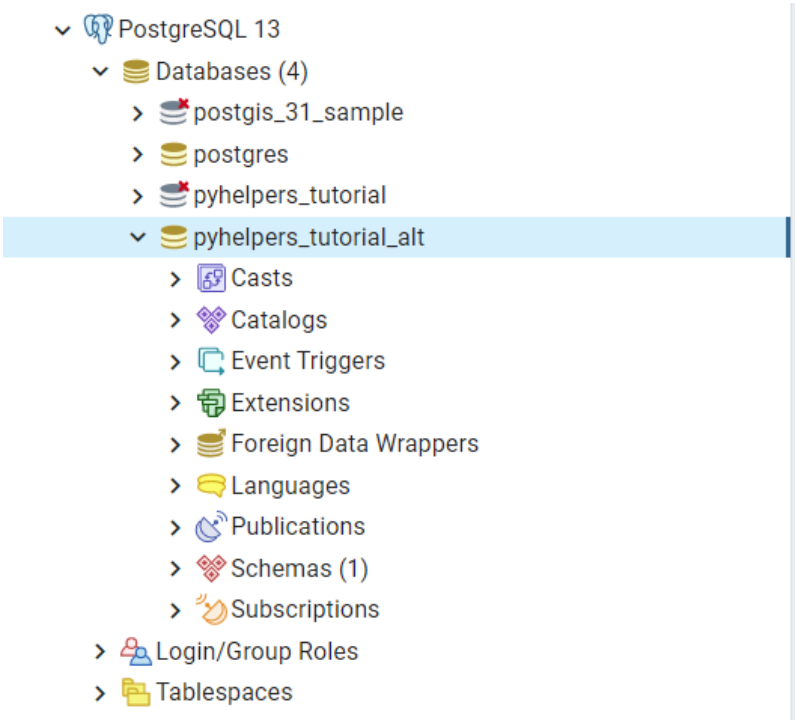

Fig. 23: The database "*pyhelpers_tutorial_alt*".

Note:

- When a new database is created, the instance `postgres` disconnects the currently-connected database and connect to the new one.

Check whether "*pyhelpers_tutorial_alt*" is the database being connected now:

```
>>> res = postgres.database_exists("pyhelpers_tutorial_alt")
>>> print(f'The database "pyhelpers_tutorial_alt" exists? {res}')
The database "pyhelpers_tutorial_alt" exists? True

>>> print(f'We are currently connected to the database "{postgres.database_name}".')
We are now connected with the database "pyhelpers_tutorial_alt".
```

To connect again to "*pyhelpers_tutorial*", we can use the method *connect_database()*:

```
>>> postgres.connect_database("pyhelpers_tutorial", 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".
```

5.8.2 Import data into a database

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

```
>>> postgres.import_data(data=data_frame, table_name="df_table", verbose=2)
To import data into "public"."df_table" at postgres:***@localhost:5432/pyhelpers_tutorial
? [No]|Yes: yes
Importing the data into the table "public"."df_table" ... Done.
```

We should now be able to see the table in pgAdmin, as illustrated in Fig. 24:

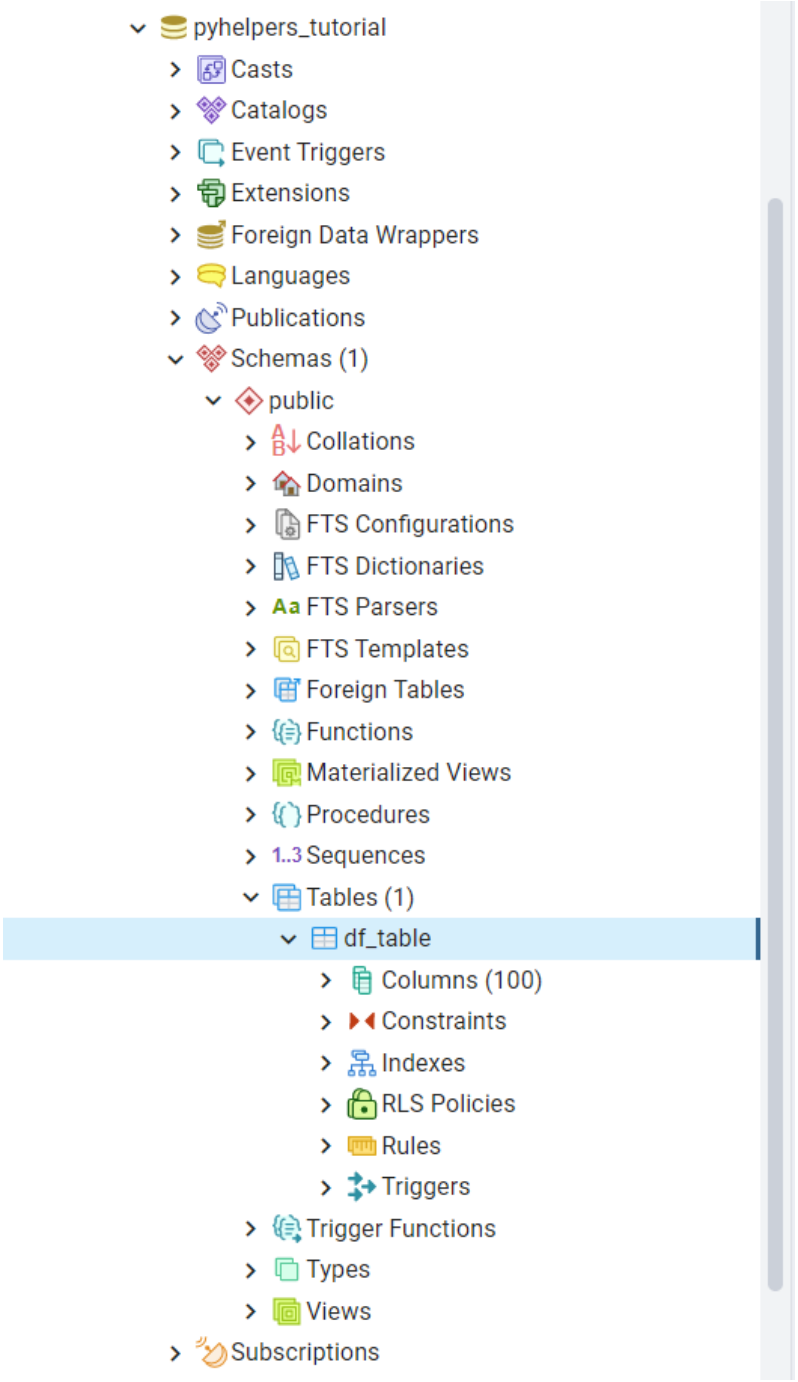

Fig. 24: The table *"public"."df_table"*.

The method *import_data()* relies on the method pandas.DataFrame.to_sql(), with the parameter `method` being set to be `'multi'` by default. Optionally, it can also take the method *psql_insert_copy()* as an argument to significantly speed up importing data into a database, especially when the data size is fairly large.

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

```
>>> postgres.import_data(
...     data=data_frame, table_name="df_table_alt", method=postgres.psql_insert_copy, verbose=2)
To import data into "public"."df_table_alt" at postgres:***@localhost:5432/pyhelpers_tutorial
? [No]|Yes: yes
Importing the data into the table "public"."df_table_alt" ... Done.
```

In pgAdmin, we can see the table has been added to the *Tables* list, as illustrated in Fig. 25:

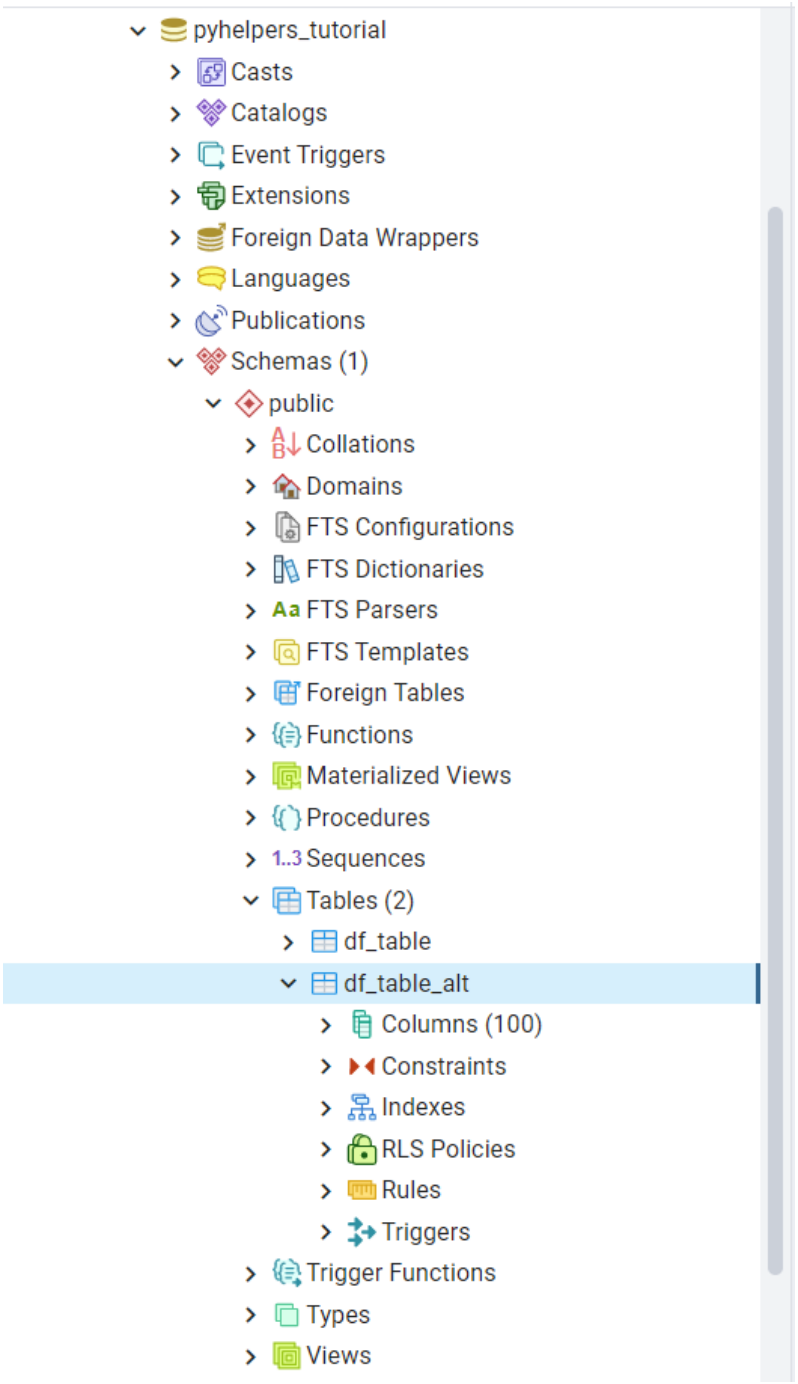

Fig. 25: The table *"public"."df_table_alt"*.

5.8.3 Fetch data from a database

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

```
>>> df_retrieval_1 = postgres.read_table(table_name="df_table")

>>> res = df_retrieval_1.equals(data_frame)
>>> print(f"`df_retrieval_1` is equal to `data_frame`? {res}")
`df_retrieval_1` is equal to `data_frame`? True
```

Alternatively, we can also use the method *read_sql_query()*, which serves as a more flexible way of reading/querying data. It takes PostgreSQL statements, and could be much faster when the queried table is fairly large. Let's try this method to fetch the same data from the table "*df_table_alt*":

```
>>> df_retrieval_2 = postgres.read_sql_query(sql_query='SELECT * FROM "public"."df_table_alt"')

>>> res = df_retrieval_2.round(8).equals(df_retrieval_1.round(8))
>>> print(f"`df_retrieval_2` is equal to `df_retrieval_1`? {res}")
`df_retrieval_2` is equal to `df_retrieval_1`? True
```

Note:

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

5.8.4 Drop 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*") by using the method *drop_table()*:

```
>>> postgres.drop_table(table_name="df_table", 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 method *drop_database()*:

```
>>> # 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="pyhelpers_tutorial_alt", verbose=True)
To drop the database "pyhelpers_tutorial_alt" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "pyhelpers_tutorial_alt" ... Done.
```

Check which database is the one being currently connected:

```
>>> 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 above, and restored the PostgreSQL server to its original status.

This is the end of the *Tutorial*.

Any issues regarding the use of the package are all welcome and should be logged/reported onto the Bug Tracker.

For more details and examples, check *Modules*.

Python Module Index

p

Index

H

I

L

M

N

O

P

R

S

T

U

V

W

X