mikeqfu / **pyhelpers**

A toolkit for facilitating data manipulation

python python3 helper-functions dir download geom settings store text postgresql sql ops

172 commits **1** branch **0** packages **12** releases Fetching contributors GPL-2.0

Python

Branch: **master**

New pull request Find file

Clone or download

Fetching latest commit...

pyhelpers	modified sql.py	May 30, 2020
tests	updated test_store.py	May 30, 2020
.gitignore	updated .gitignore	May 30, 2020
LICENSE	Create LICENSE	Aug 30, 2019
README.md	updated README.md	May 30, 2020
requirements.txt	updated requirements.txt	May 30, 2020
setup.py	updated setup.py	May 30, 2020

README.md

pyhelpers

A toolkit for facilitating data manipulation

Author: Qian Fu Follow

PyPI v1.1.2 Python 3 License GPL-2.0 Code size 118 kB Downloads 478/month

Contents

Installation

```
pip install pyhelpers
```

Note:

- Only a few frequently-used dependencies are installed through `pip install`.
- When importing the module/functions whose dependencies are not available with the installation (or if you happen not to have those dependencies installed yet), an "*ModuleNotFoundError*" will be prompted and you may install them separately.

Quick start - some examples

The current version includes the following modules:

- `settings`
- `dir`
- `download`
- `store`
- `geom`
- `text`
- `ops`
- `sql`

Each of the modules includes a number of functions. For a quick start and demonstration purposes, only a couple of examples are provided below.

settings

This module can be used to change some common settings with pandas, numpy, matplotlib and gdal.

```
from pyhelpers.settings import pd_preferences
```

The function `pd_preferences()` changes a few default pandas settings (when `reset=False`), such as the display representation and maximum number of columns when displaying a `pandas.DataFrame`.

```
pd_preferences(reset=False)
```

If `reset=True`, all changed parameters are reset to their default values. Note that the preset parameters are for the module creator's preference; you can always change them in the source code to whatever suits your use.

dir

This module can be used to help manipulate directories.

```
from pyhelpers.dir import cd
```

The function `cd()` returns the current working directory

```
print(cd())
```

If you would like to direct to a customised folder, say `"test_dir"`, use `cd()` to change directory:

```
path_to_folder = cd("test_dir", mkdir=False)
print(path_to_folder)
```

If the directory `path_to_folder` does not exist, setting `mkdir=True` (default: `False`), i.e. `cd("test_dir", mkdir=True)` will create it.

More examples: (You could see the difference between `path_to_pickle` and `path_to_test_pickle` below)

```
path_to_pickle = cd("test_dir", "dat.pickle")
print(path_to_pickle)
```

```
path_to_test_pickle = cd("test_dir", "data", "dat.pickle")
# or, path_to_test_pickle == cd("test_dir\\data\\dat.pickle")
print(path_to_test_pickle)
```

Another function, `regulate_input_data_dir()`, may also be helpful sometimes:

```
from pyhelpers.dir import regulate_input_data_dir

print(regulate_input_data_dir("test_dir"))
print(regulate_input_data_dir(path_to_test_pickle))
```

download

Note that this module requires requests and tqdm.

```
from pyhelpers.download import download
```

Suppose you would like to download a Python logo from online where URL is as follows:

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

Firstly, specify where the .png file will be saved and what the filename is. For example, to name the downloaded file as `"python-logo.png"` and save it to a folder named `"images"`:

```
python_logo_dir = cd("test_dir", "images")
path_to_python_logo = cd(python_logo_dir, "python-logo.png")
```

Then use `download()`

```
download(url, path_to_python_logo)
```

You may view the downloaded picture by using Pillow:

```
from PIL import Image

python_logo = Image.open(path_to_python_logo)
python_logo.show()
```

If you would like to remove the download directory, "../test_dir/images":

```
from pyhelpers.dir import rm_dir
```

By setting `confirmation_required=True` (default), you will be asked to confirm whether you want to proceed if the folder is not empty:

```
# Remove "picture" folder
rm_dir(cd("test_dir\\images"), confirmation_required=True)
```

store

This module can be used to help save and retrieve data. Note that some functions require openpyxl, XlsxWriter and xlrd.

Let's create a `pandas.DataFrame` first by using the above `xy_array`:

```
import numpy as np
import pandas as pd

xy_array = np.array([(530034, 180381),   # London
                     (406689, 286822),   # Birmingham
                     (383819, 398052),   # Manchester
                     (582044, 152953)],  # Leeds
                    dtype=np.int64)

dat = pd.DataFrame(xy_array, columns=['Easting', 'Northing'])
```

If you would like to save `dat` as a pickle file and retrieve it later, use `save_pickle` and `load_pickle`:

```
from pyhelpers.store import save_pickle, load_pickle
```

To save `dat` to `path_to_test_pickle` (see `dir`):

```
save_pickle(dat, path_to_test_pickle, verbose=True)  # default: verbose=False
```

To retrieve/load `dat` from `path_to_test_pickle`:

```
dat_retrieved = load_pickle(path_to_test_pickle, verbose=True)
```

`dat_retrieved` and `dat` should be identical:

```
print(dat_retrieved.equals(dat))  # True
```

In addition, `store.py` also have functions for working with some other formats, such as

- `save_json()` and `load_json()` for .json
- `save_spreadsheet()` for .csv and .xlsx/.xls
- `save_multiple_spreadsheets()` and `load_multiple_spreadsheets()` for .xlsx/.xls
- `save_feather()` and `load_feather()` for .feather
- ...

geom

This module can be used to help data manipulation related to geometric data and geographical data.

For example, if you need to convert coordinates from OSGB36 (British national grid) to WGS84 (latitude and longitude), use `osgb36_to_wgs84`:

```
from pyhelpers.geom import osgb36_to_wgs84
```

To convert a single coordinate, `xy`:

```
xy = np.array((530034, 180381))  # London

easting, northing = xy
lonlat = osgb36_to_wgs84(easting, northing)  # osgb36_to_wgs84(xy[0], xy[1])

print(lonlat)  # (-0.12772400574286874, 51.50740692743041)
```

To convert an array of OSGB36 coordinates, `xy_array`:

```
eastings, northings = xy_array.T
lonlat_array = np.array(osgb36_to_wgs84(eastings, northings))

print(lonlat_array.T)
# [[-0.12772401 51.50740693]
#  [-1.90294064 52.47928436]
#  [-2.24527795 53.47894006]
#  [ 0.60693267 51.24669501]]
```

Similarly, if you would like to convert coordinates from latitude/longitude (WGS84) to easting/northing (OSGB36), import `wgs84_to_osgb36` instead.

text

This module can be used to help manipulate text data.

For example, suppose you have a `str` type variable, `string`:

```
string = 'ang'
```

If you would like to find the most similar text to one of the following `lookup_list`:

```
lookup_list = ['Anglia',
               'East Coast',
               'East Midlands',
               'North and East',
               'London North Western',
               'Scotland',
               'South East',
               'Wales',
               'Wessex',
               'Western']
```

Try the function `find_similar_str()`:

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

Setting `processor='fuzzywuzzy'` requires fuzzywuzzy (recommended) - `token_set_ratio`

```
result_1 = find_similar_str(string, lookup_list, processor='fuzzywuzzy')

print(result_1)  # Anglia
```

Setting `processor='nltk'` requires nltk - `edit_distance`

```
result_2 = find_similar_str(string, lookup_list, processor='nltk',
                            substitution_cost=1)

print(result_2)  # Anglia
```

ops

This module gathers miscellaneous functions.

For example, if you would like to request a confirmation before proceeding with some processes, use `confirmed`:

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

You may specify, by setting `prompt`, the prompting message as to the confirmation:

```
confirmed(prompt="Continue?...", confirmation_required=True)
# Continue?... [No]|Yes: # Type something
```

If you input `Yes` (or `Y`, `yes`, or something like `ye`), it should return `True`; otherwise, `False` (if the input being `No` or something like `n`). When `confirmation_required=False`, meaning that a confirmation is not required, this function would be null as it will just return `True`.

sql

This module provides a convenient way to establish a connection with a SQL server. The current version supports only PostgreSQL and a quick example is given below.

```
from pyhelpers.sql import PostgreSQL
```

(1) Connect to a database

Now you can connect a PostgreSQL server by specifying *host* (default: `'localhost'`), *port* (default: `5432`), *username* (default: `'postgres'`), *password* and *name of the database* (default: `'postgres'`)

```
# Connect to 'postgres' (using all default parameters)
testdb = PostgreSQL(host='localhost', port=5432, username='postgres',
                    password=None, database_name='postgres', verbose=True)
```

As `password=None`, you will be asked to input your password. And if `host`, `port`, `username` and `database_name` are all `None`, you will be asked to input manually as you run `PostgreSQL()`.

(2) Import and dump data into the database

After you have successfully established the connection, you could try to dump `dat` (see above) into the database "postgres":

```
testdb.dump_data(dat, table_name='test_table', schema_name='public',
                 if_exists='replace', force_replace=False,
                 chunk_size=None, col_type=None, method='multi',
                 verbose=True)
```

The method `.dump_data()` relies on `pandas.DataFrame.to_sql()`; however, the default `method` is set to be `'multi'` (i.e. `method='multi'`) for a faster process. In addition, `.dump_data()` further includes a callable `psql_insert_copy`, whereby the processing speed could be even faster.

For example:

```
testdb.dump_data(dat, table_name='test_table',
                 method=testdb.psql_insert_copy, verbose=True)
```

(3) Read data from the database

To retrieve the dumped data, use the method `.read_table()`:

```
dat_retrieved = testdb.read_table('test_table')

print(dat.equals(dat_retrieved))  # True
```

Besides, there is an alternative way, which is more flexible with PostgreSQL statement (and could be faster especially when the table is fairly large):

```
sql_query = 'SELECT * FROM test_table'  # Or 'SELECT * FROM public.test_table'
dat_retrieved_ = testdb.read_sql_query(sql_query)
```

Note that `sql_query` should end without ';'

```
print(dat.equals(dat_retrieved_))  # True
```

(4) Some other methods

To drop the table **'test_table'**, if `confirmation_required=True` (default: `False`), you will be asked to confirm whether you are sure to drop the table:

```
testdb.drop_table('test_table', schema_name='public',
                  confirmation_required=True, verbose=True)
# Confirmed to drop the table ... ? [No]|Yes: # Type something
# The table "test_table" has been dropped successfully.
```

If you would like to create your own database and name it as `'test_database'`:

```
testdb.create_database("test_database", verbose=True)
# Creating a database "test_database" ... Done.
```

After successfully creating the database, you could check:

```
test_db_exists = testdb.database_exists("test_database")
print(test_db_exists)  # True
```

```
test_db_name = testdb.database_name
print(test_db_name)  # test_database
```

To drop this database, use the method `.drop_database()`. Setting `confirmation_required=True` (default and recommended) would require you to confirm whether you are sure to proceed:

```
testdb.drop_database(confirmation_required=True, verbose=True)
# Confirmed to drop the database ... ? [No]|Yes: # Type something
# Dropping the database "test_database" ... Done.
```