PyDriosm Documentation

Release 2.0.3

Qian Fu

Apr 25, 2021

CONTENTS

CHAPTER

ONE

INSTALLATION

To install the latest release of PyDriosm at PyPI via pip:

```
pip install --upgrade pydriosm
```

To install the more recent version hosted directly from GitHub repository:

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

Note: Possibilities of `pip install` being unsuccessful or causing errors:

- **For *Windows* users:** The `pip` method might fail to install some dependencies, such as GDAL, Fiona and Shapely. If errors occur when directly installing any of those dependencies, `pip install` instead their respective *.whl* files, which can be downloaded from Unofficial Windows Binaries for Python Extension Packages. After the *.whl* files are installed successfully, try `pip install pydriosm` again.
- **For *Linux/Unix* users:** To try out any earlier version (<2.0.0) that is not compatible with 2.0.0+, check this page for instructions if errors occur during installation.

To test if PyDriosm is correctly installed, try to import the package via an interpreter shell:

```
>>> import pydriosm

>>> pydriosm.__version__
```

```
The current release version is: 2.0.3
```

Note:

- If using a virtual environment, ensure that it is activated.
- To ensure you get the most recent version, it is always recommended to add `--upgrade` (or `-U`) to `pip install`.
- The package has not yet been tested with Python 2. For users who have installed both Python 2 and Python 3, it would be recommended to replace `pip` with `pip3`. But you are more than welcome to volunteer testing the package with Python 2 and any issues should be logged/reported onto the Issues page.
- For more general instructions, check the Installing Packages page.

CHAPTER

TWO

QUICK START

For a demonstration of how pydriosm works with OpenStreetMap (OSM) data, this part of the documentation provides a quick guide with some practical examples of using the package to download, parse and store the OSM data.

Note:

- All the data for this quick-start tutorial will be downloaded and saved to a directory named "tests" (which will be created if it does not exist) at the current working directory as you move from one code block to another.
- The downloaded data and those being generated during the tutorial will all be deleted from the "tests" directory; a manual confirmation will be prompted at the end of the tutorial to determine whether the "tests" folder should remain.

2.1 Download data

The current release version of the package works mainly for the OSM data extracts that is available for free download from Geofabrik and BBBike download servers.

To start with, you could use the class *GeofabrikDownloader* (see also *pydriosm.downloader*) to get a sample from the free Geofabrik download server.

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> # Create an instance for downloading the Geofabrik data extracts
>>> geofabrik_downloader = GeofabrikDownloader()
```

To explore what data is available for download, you may check out a download catalogue by using the method *.get_download_catalogue()* :

```
>>> # A download catalogue for all subregions
>>> geofabrik_download_catalogue = geofabrik_downloader.get_download_catalogue()

>>> # Check the column names
>>> geofabrik_download_catalogue.columns.tolist()
['Subregion',
 'SubregionURL',
 '.osm.pbf',
 '.osm.pbf.Size',
 '.shp.zip',
 '.osm.bz2']
```

(continues on next page)

(continued from previous page)

```
>>> geofabrik_download_catalogue.head()
      Subregion  ...                                              .osm.bz2
0       Algeria  ...  https://download.geofabrik.de/africa/algeria-l...
1        Angola  ...  https://download.geofabrik.de/africa/angola-la...
2         Benin  ...  https://download.geofabrik.de/africa/benin-lat...
3      Botswana  ...  https://download.geofabrik.de/africa/botswana-...
4  Burkina Faso  ...  https://download.geofabrik.de/africa/burkina-f...
[5 rows x 6 columns]
```

If you'd like to download say the protocolbuffer binary format (PBF) data of a specific geographic region, you need to specify the name of the region and file format (e.g. `".pbf"`). For example, to download the PBF data of `'London'` and save it to a local directory named `"tests"`:

```
>>> subregion_name = 'London'  # case-insensitive
>>> osm_file_format = ".pbf"  # or ".osm.pbf"
>>> download_dir = "tests"  # a download directory

>>> # Download the OSM PBF data of London from Geofabrik
>>> geofabrik_downloader.download_osm_data(subregion_name, osm_file_format,
...                                        download_dir, verbose=True)
To download .osm.pbf data of the following geographic region(s):
    Greater London
? [No]|Yes: yes
Downloading "greater-london-latest.osm.pbf" to "tests\" ... Done.
```

Note:

- If the data file does not exist at the specific directory, you'll be asked to confirm whether to proceed to download it, as a function parameter `confirmation_required` is `True` by default. To skip the confirmation, you just need to set it to be `False`.
- If the `download_dir` is `None` by default, the downloaded data file would be saved to a default data directory, which in this case should be `"osm_geofabrik\Europe\Great Britain\England\"`.

Now you should be able to find the downloaded data file at *"<current working directory>tests*"*, *and the filename is "*greater-london-latest.osm.pbf"* by default.

To retrieve the default filename and the full path to the downloaded file, you could set the parameter `ret_download_path` to be `True` when executing the method:

```
>>> path_to_london_pbf = geofabrik_downloader.download_osm_data(
...     subregion_name, osm_file_format, download_dir, confirmation_required=False,
...     ret_download_path=True)

>>> import os

>>> # Default filename:
>>> london_pbf_filename = os.path.basename(path_to_london_pbf)
>>> print(f"Default filename: \"{london_pbf_filename}\"")
Default filename: "greater-london-latest.osm.pbf"

>>> # Relative file path:
```

(continues on next page)

(continued from previous page)

```
>>> print(f"Current (relative) file path: \"{os.path.relpath(path_to_london_pbf)}\"")
Current (relative) file path: "tests\greater-london-latest.osm.pbf"
```

Alternatively, you could also make use of the method *.get_default_path_to_osm_file()* to get the default path to the data file (even when it does not exist):

```
>>> london_pbf_filename, default_path_to_london_pbf = \
...     geofabrik_downloader.get_default_path_to_osm_file(subregion_name, osm_file_format)

>>> print(f"Default filename: \"{london_pbf_filename}\"")
Default filename: "greater-london-latest.osm.pbf"

>>> path_to_london_pbf = os.path.join(download_dir, london_pbf_filename)
>>> print(f"Current (relative) file path: \"{os.path.relpath(path_to_london_pbf)}\"")
Current (relative) file path: "tests\greater-london-latest.osm.pbf"
```

In addition, you can also download data of multiple (sub)regions at one go. For example, to download PBF data of three different regions, including `'Rutland'`, `'West Yorkshire'` and `'West Midlands'` (where you can set `confirmation_required=False` to waive the requirement of confirmation to proceed to download the data):

```
>>> subregion_names = ['Rutland', 'West Yorkshire', 'West Midlands']

>>> paths_to_pbf = geofabrik_downloader.download_osm_data(
...     subregion_names, osm_file_format, download_dir, ret_download_path=True,
...     verbose=True)
To download .osm.pbf data of the following geographic region(s):
    Rutland
    West Yorkshire
    West Midlands
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "tests\" ... Done.
Downloading "west-yorkshire-latest.osm.pbf" to "tests\" ... Done.
Downloading "west-midlands-latest.osm.pbf" to "tests\" ... Done.

>>> type(paths_to_pbf)
list

>>> for path_to_pbf in paths_to_pbf:
...     print(f"\"{os.path.relpath(path_to_pbf)}\"")
"tests\rutland-latest.osm.pbf"
"tests\west-yorkshire-latest.osm.pbf"
"tests\west-midlands-latest.osm.pbf"
```

2.2 Read/parse data

To read/parse any of the downloaded data files above, you could use the class *GeofabrikReader* (see also *pydriosm.reader*).

```
>>> # Create an instance for reading the downloaded Geofabrik data extracts
>>> from pydriosm.reader import GeofabrikReader

>>> geofabrik_reader = GeofabrikReader()
```

2.2.1 PBF data (.pbf / .osm.pbf)

To read the PBF data, you can use the method *.read_osm_pbf()*, whose parser depends largely on GDAL/OGR. Also check out the function `parse_osm_pbf()` for more details.

Now, let's try to read the PBF data of Rutland:

```
>>> subregion_name = 'Rutland'
>>> data_dir = download_dir  # "tests"

>>> rutland_pbf_raw = geofabrik_reader.read_osm_pbf(subregion_name, data_dir)

>>> type(rutland_pbf_raw)
dict
```

`rutland_pbf_raw` is in dict type and has five keys: `'points'`, `'lines'`, `'multilinestrings'`, `'multipolygons'` and `'other_relations'`, corresponding to the names of the five different layers of the PBF data.

Check out the **'points'** layer:

```
>>> rutland_pbf_points = rutland_pbf_raw['points']

>>> rutland_pbf_points.head()
                                              points
0  {"type": "Feature", "geometry": {"type": "Poin...
1  {"type": "Feature", "geometry": {"type": "Poin...
2  {"type": "Feature", "geometry": {"type": "Poin...
3  {"type": "Feature", "geometry": {"type": "Poin...
4  {"type": "Feature", "geometry": {"type": "Poin...
```

Each row of `rutland_pbf_points` is textual GeoJSON data, which is a nested dictionary.

```
>>> import json

>>> rutland_pbf_points_0 = rutland_pbf_points['points'][0]
>>> type(rutland_pbf_points_0)
str

>>> # Decode the str-type data
>>> rutland_pbf_points_0_ = json.loads(rutland_pbf_points_0)
>>> type(rutland_pbf_points_0_)
dict

>>> list(rutland_pbf_points_0_.keys())
['type', 'geometry', 'properties', 'id']

>>> rutland_pbf_points_0_
{'type': 'Feature',
 'geometry': {'type': 'Point', 'coordinates': [-0.5134241, 52.6555853]},
 'properties': {'osm_id': '488432',
  'name': None,
  'barrier': None,
  'highway': None,
  'ref': None,
  'address': None,
  'is_in': None,
  'place': None,
  'man_made': None,
  'other_tags': '"odbl"=>"clean"'},
 'id': 488432}
```

The charts (Fig. 1 - Fig. 5) below illustrate the different geometry types and structures (i.e. all keys within the corresponding GeoJSON data) for each layer:

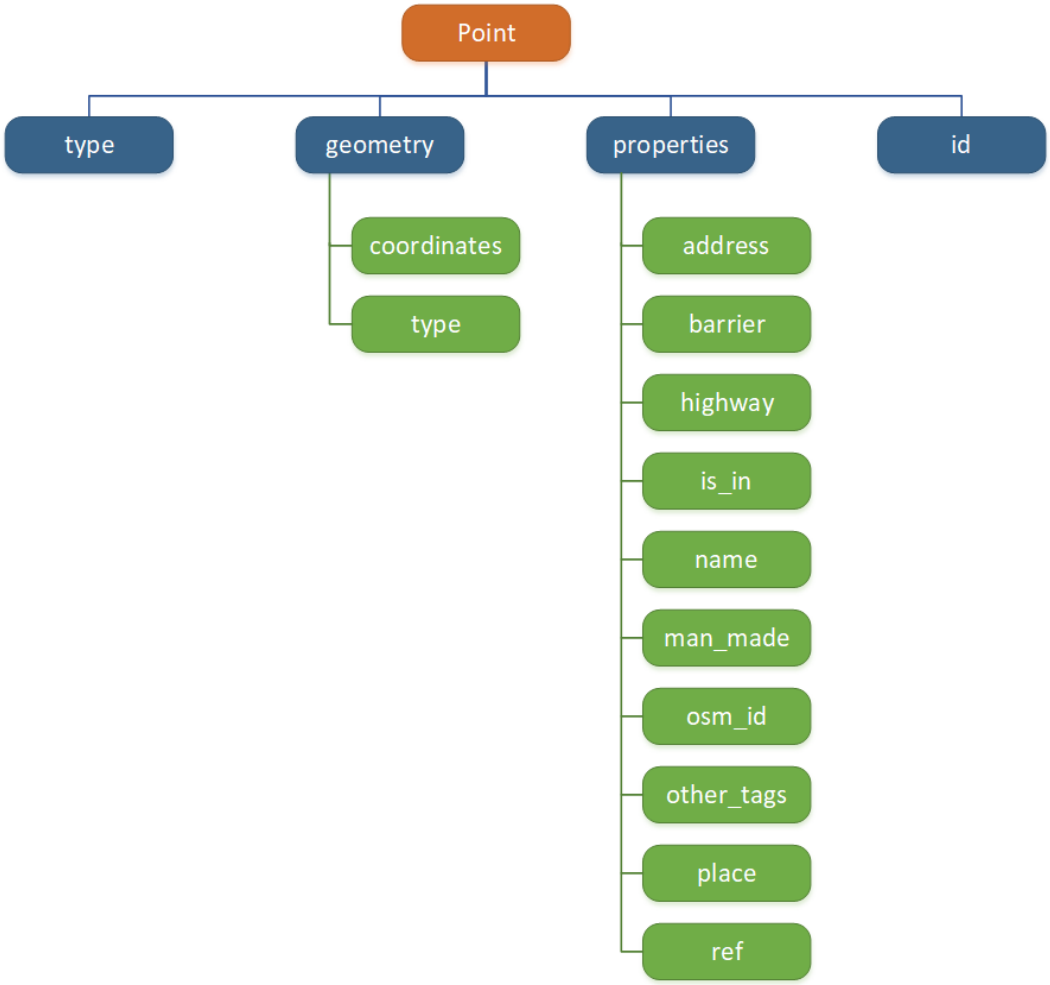

Fig. 1: Type of the geometry object and keys within the nested dictionary of `'points'`.

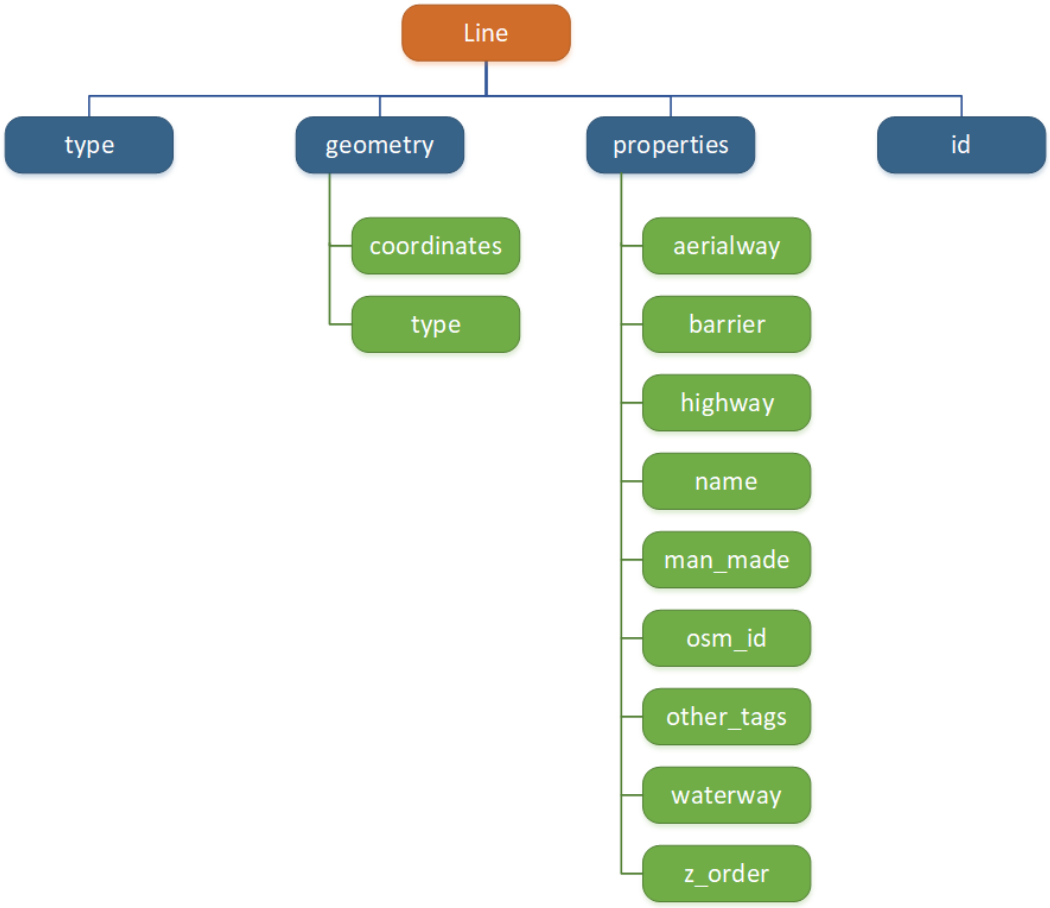

Fig. 2: Type of the geometry object and keys within the nested dictionary of `'lines'`.

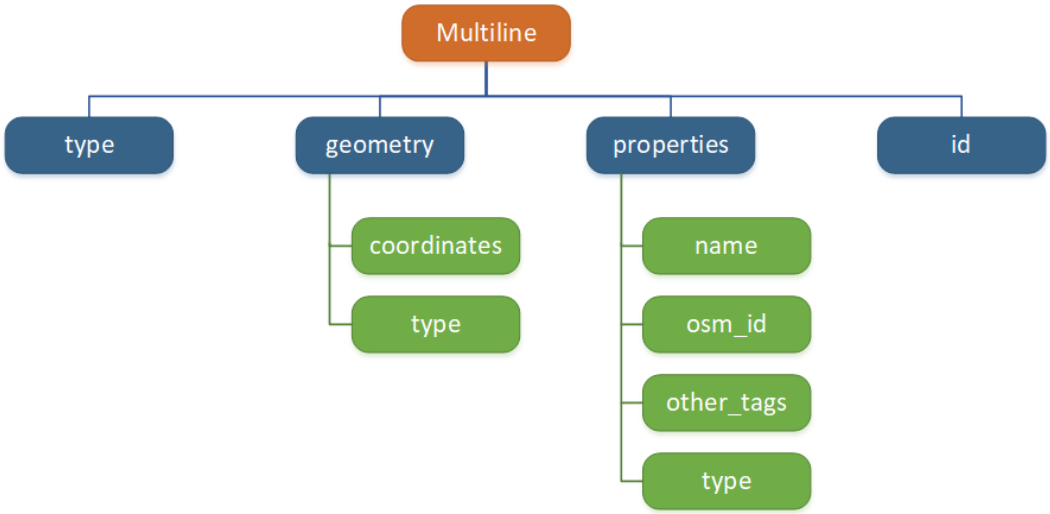

Fig. 3: Type of the geometry object and keys within the nested dictionary of `'multilinestrings'`.

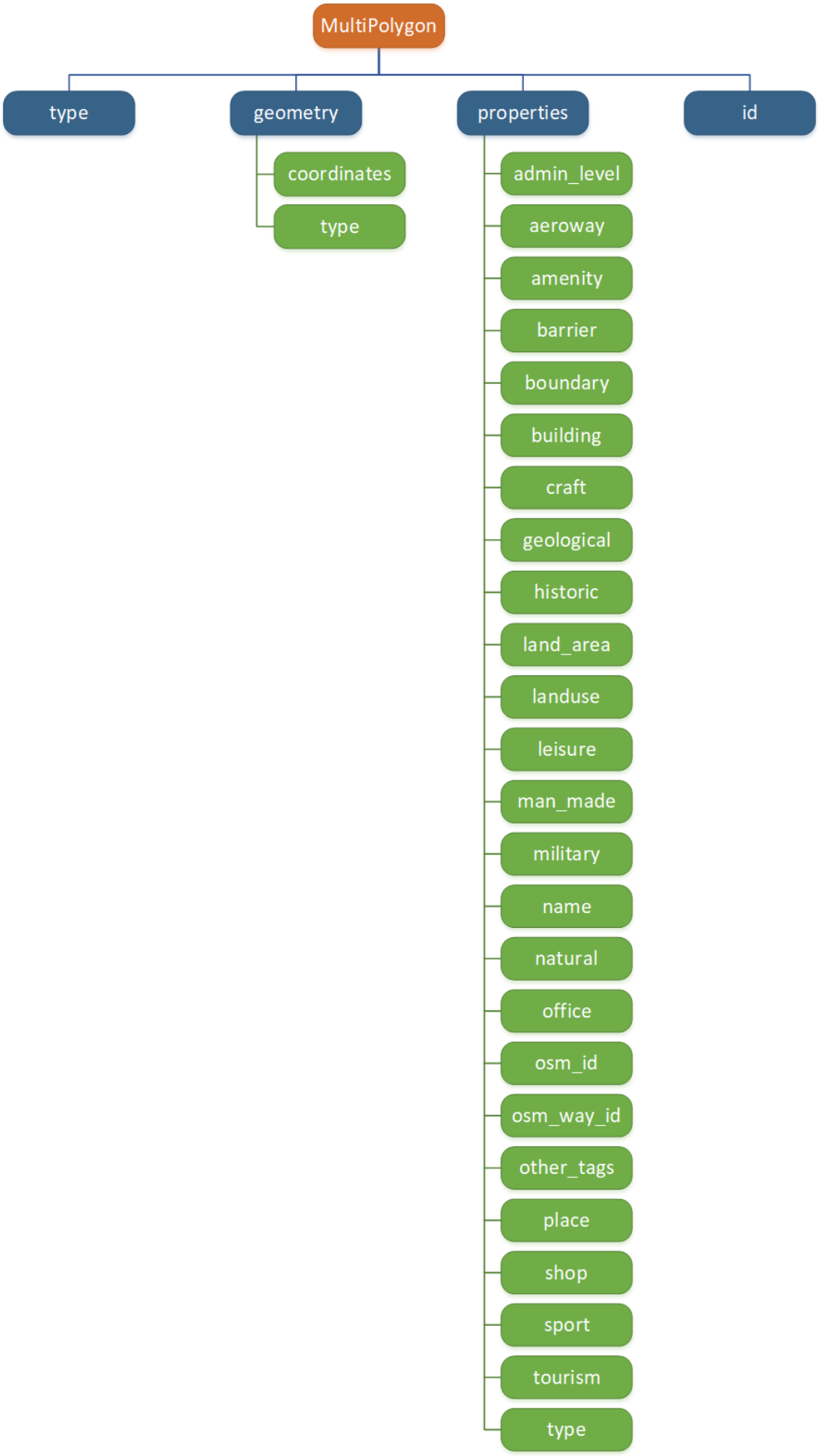

Fig. 4: Type of the geometry object and keys within the nested dictionary of `'multipolygons'`.

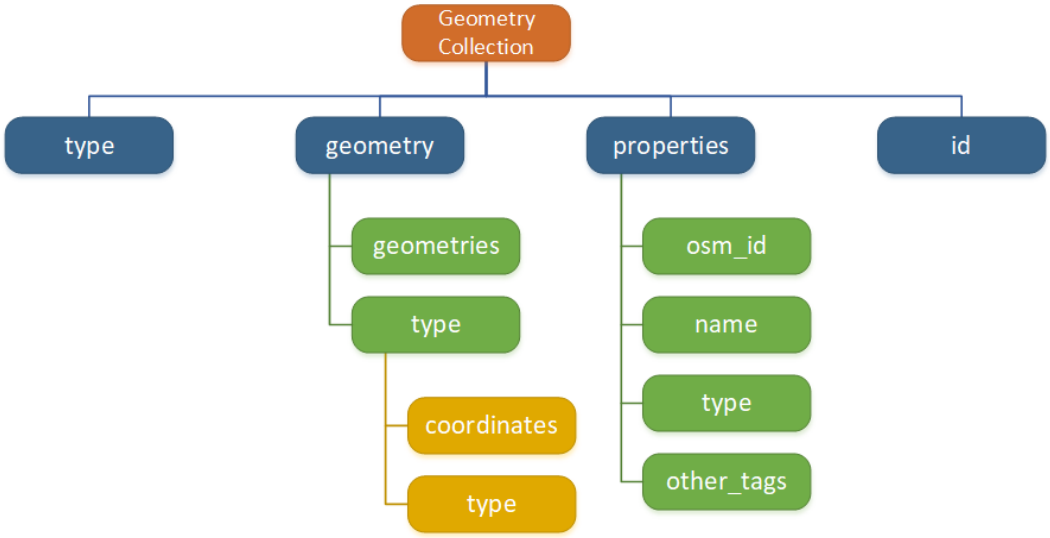

Fig. 5: Type of the geometry object and keys within the nested dictionary of `'other_relations'`.

If you set `parse_raw_feat` (which defaults to `False`) to be `True` when reading the PBF data, you can also parse the GeoJSON record to obtain data of 'visually' (though not virtually) higher level of granularity:

```
>>> rutland_pbf_parsed = geofabrik_reader.read_osm_pbf(subregion_name, data_dir,
...                                                    parse_raw_feat=True,
...                                                    verbose=True)
Parsing "\tests\rutland-latest.osm.pbf" ... Done.

>>> # Data of the parsed 'points' layer
>>> rutland_pbf_parsed_points = rutland_pbf_parsed['points']

>>> rutland_pbf_parsed_points.head()
         id              coordinates  ... man_made                   other_tags
0    488432  [-0.5134241, 52.6555853]  ...     None               "odbl"=>"clean"
1    488658  [-0.5313354, 52.6737716]  ...     None                          None
2  13883868  [-0.7229332, 52.5889864]  ...     None                          None
3  14049101  [-0.7249816, 52.6748426]  ...     None  "traffic_calming"=>"cushion"
4  14558402  [-0.7266581, 52.6695058]  ...     None      "direction"=>"clockwise"
[5 rows x 12 columns]
```

Note:

- The data can be further transformed/parsed through two more parameters, `transform_geom` and `transform_other_tags`, both of which default to `False`.
- The method *`.read_osm_pbf()`* may take dozens of minutes or longer to parse large-size PBF data file. If the size of a data file is greater than a specified `chunk_size_limit` (which defaults to 50 MB), the data will be parsed in a chunk-wise manner.
- If only the name of a geographic region is provided, e.g. `rutland_pbf = geofabrik_reader.read_osm_pbf(subregion_name='London')`, the function will go to look for the data file at the default file path. Otherwise, you must specify `data_dir` where the data file is located.
- If the data file does not exist at the default or a specified directory, the function will try to download it first. By default, a manual confirmation of downloading the data is required.

To waive the requirement, set `download_confirmation_required=False`.

- If `pickle_it=True`, the parsed data will be saved as a Pickle file. The function will try to load the Pickle file next time when you run it, provided that `update=False` (default); if `update=True`, the function will try to download and parse the latest version of the data file.

2.2.2 Shapefiles (.shp.zip / .shp)

To read shapefile data, you can use the method *`.read_shp_zip()`*, which depends on pyshp (or optionally, GeoPandas, which is not required for the installation of PyDriosm).

For example, let's try to read the 'railways' layer of the shapefile data of London:

```
>>> subregion_name = 'London'
>>> layer_name = 'railways'  # if layer_name=None (default), all layers will be included

>>> london_shp = geofabrik_reader.read_shp_zip(subregion_name, layer_names=layer_name,
...                                            feature_names=None, data_dir=data_dir,
...                                            verbose=True)
To download .shp.zip data of the following geographic region(s):
    Greater London
? [No]|Yes: yes
Downloading "greater-london-latest-free.shp.zip" to "tests\" ... Done.
Extracting the following layer(s):
    'railways'
from "tests\greater-london-latest-free.shp.zip" ...
to "tests\greater-london-latest-free-shp\"
Done.
```

`london_shp` is in dict type, with the default `layer_name` being its key.

```
>>> london_railways_shp = london_shp[layer_name]

>>> london_railways_shp.head()
   osm_id  code  ...                                        coordinates shape_type
0   30804  6101  ...  [(0.0048644, 51.6279262), (0.0061979, 51.62926...          3
1  101298  6103  ...  [(-0.2249632, 51.4935445), (-0.2250662, 51.494...          3
2  101486  6103  ...  [(-0.2055497, 51.5195429), (-0.2051377, 51.519...          3
3  101511  6101  ...  [(-0.2119027, 51.5241906), (-0.2108059, 51.523...          3
4  282898  6103  ...  [(-0.1862586, 51.6159083), (-0.1868721, 51.613...          3
[5 rows x 9 columns]
```

Note:

- The parameter `feature_names` is related to `'fclass'` in `london_railways_shp`. You can specify one feature name (or multiple feature names) to get a subset of `london_railways_shp`.
- Similar to the method *`.read_osm_pbf()`*, if *`.read_shp_zip()`* could not find the target *.shp* file at the default or specified directory (i.e. `data_dir`), it will try to extract the *.shp* file from the *.shp.zip* file (or download the *.shp.zip* file first if it does not exist, in which case a confirmation to proceed is by default required as `download_confirmation_required=True`).

- If you'd like to delete the *.shp* files and/or the downloaded data file (ending with *.shp.zip*), set the parameters `rm_extracts=True` and/or `rm_shp_zip=True`.

In addition, you can use the method *.merge_subregion_layer_shp()* to merge multiple shapefiles of different subregions over a specific layer.

For example, to merge the 'railways' layer of London and Kent:

```
>>> layer_name = 'railways'
>>> subregion_names = ['London', 'Kent']

>>> path_to_merged_shp = geofabrik_reader.merge_subregion_layer_shp(
...     subregion_names, layer_name, data_dir, verbose=True, ret_merged_shp_path=True)
"greater-london-latest-free.shp.zip" is already available at "tests\".
To download .shp.zip data of the following geographic region(s):
    Kent
? [No]|Yes: >? yes
Downloading "kent-latest-free.shp.zip" to "tests\" ... Done.
Extracting the following layer(s):
    'railways'
from "tests\greater-london-latest-free.shp.zip" ...
to "tests\greater-london-latest-free-shp\"
Done.
Extracting the following layer(s):
    'railways'
from "tests\kent-latest-free.shp.zip" ...
to "tests\kent-latest-free-shp\"
Done.
Merging the following shapefiles:
    "greater-london_gis_osm_railways_free_1.shp"
    "kent_gis_osm_railways_free_1.shp"
In progress ... Done.
Find the merged shapefile at "tests\greater-london_kent_railways\".

>>> # Relative path of the merged shapefile
>>> print(os.path.relpath(path_to_merged_shp))
tests\greater-london_kent_railways\greater-london_kent_railways.shp
```

For more details, also check out the functions: *merge_shps()* and *merge_layer_shps()*.

2.3 Import and fetch data with a PostgreSQL server

In addition to downloading and reading OSM data, the package further provides a module *pydriosm.ios* for communicating with PostgreSQL server, that is, to import the OSM data into, and fetch it from, PostgreSQL databases.

To establish a connection with the server, you need to specify the username, password, host address of a PostgreSQL server and name of a database. For example:

```
>>> from pydriosm.ios import PostgresOSM

>>> host = 'localhost'
>>> port = 5432
>>> username = 'postgres'
>>> password = None  # You need to type it in manually if `None`
>>> database_name = 'osmdb_test'
```

(continues on next page)

(continued from previous page)

```
>>> # Create an instance of a running PostgreSQL server
>>> osmdb_test = PostgresOSM(host, port, username, password, database_name)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
```

The example is illustrated in Fig. 6:

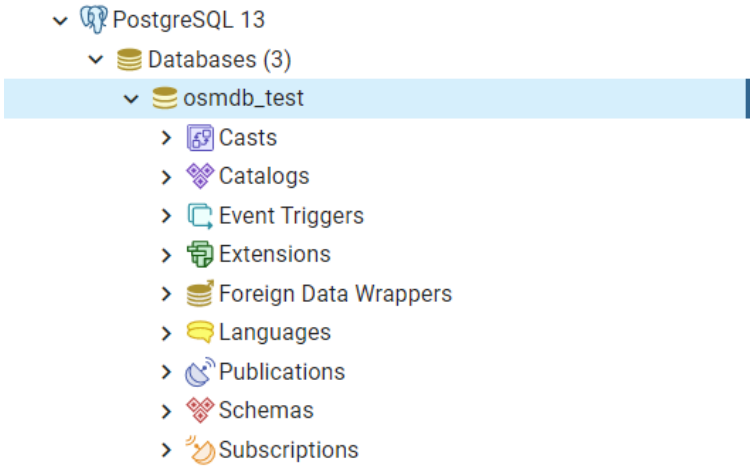

Fig. 6: An illustration of the database named *'osmdb_test'*.

Note:

- If you don't specify a password (for creating the instance `osmdb_test`) as the parameter `password` is `None` by default, you'll be asked to manually type in the password to the PostgreSQL server.
- The class *`PostgresOSM`* has incorporated all available classes from the modules: *`downloader`* and *`reader`* as properties. In the case of the above instance, `osmdb_test.Downloader` is equivalent to the class *`GeofabrikDownloader`*, as the parameter `data_source` is `'Geofabrik'` by default.
- To relate the instance `osmdb_test` to 'BBBike' data, you could: 1) recreate an instance by setting `data_source='BBBike'`; or 2) set `osmdb_test.DataSource` to be `'BBBike'`

2.3.1 Import data into the database

To import any of the above OSM data to a database in the connected PostgreSQL server, you can use the method *`.import_osm_data()`* or *`.import_subregion_osm_pbf()`*.

For example, let's now try to import `rutland_pbf_parsed` that you have obtained from *PBF data (.osm.pbf / .pbf)*:

```
>>> subregion_name = 'Rutland'

>>> osmdb_test.import_osm_data(rutland_pbf_parsed, table_name=subregion_name, verbose=True)
To import data into table "Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
```

(continues on next page)

(continued from previous page)

```
    "points" ... Done: <total of rows> features.
    "lines" ... Done: <total of rows> features.
    "multilinestrings" ... Done: <total of rows> features.
    "multipolygons" ... Done: <total of rows> features.
    "other_relations" ... Done: <total of rows> features.
```

Note: The parameter `schema_names` is `None` by default, meaning that you are going to import all the five layers of the PBF data into the database.

In the example above, five schemas, including 'points', 'lines', 'multilinestrings', 'multipolygons' and 'other_relations' are, if they do not exist, created in the database 'osmdb_test'. Each of the schemas corresponds to a key (i.e. name of a layer) of `rutland_pbf_parsed` (as illustrated in Fig. 7); and the data of each layer is imported into a table named as 'Rutland' under the corresponding schema (as illustrated in Fig. 8).

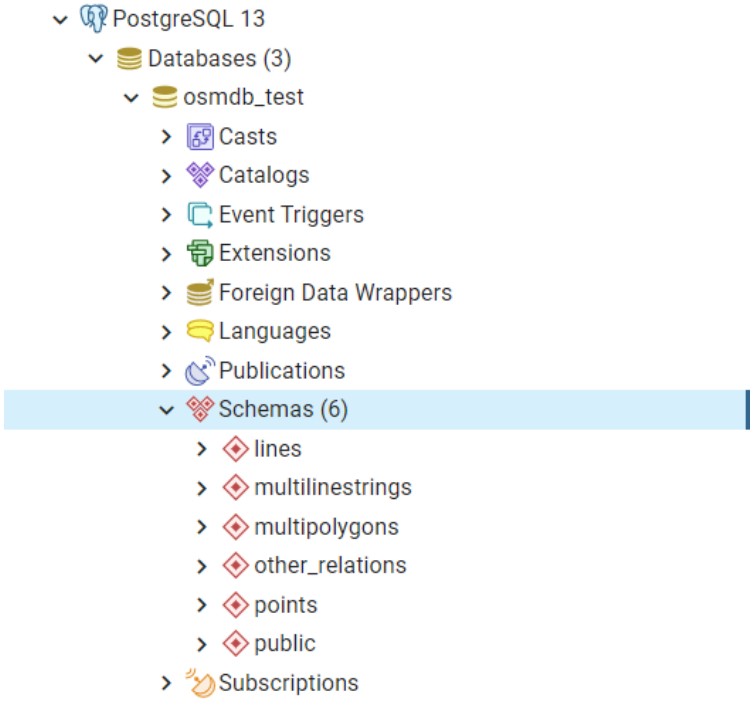

Fig. 7: An illustration of schemas for importing OSM PBF data into a PostgreSQL database.

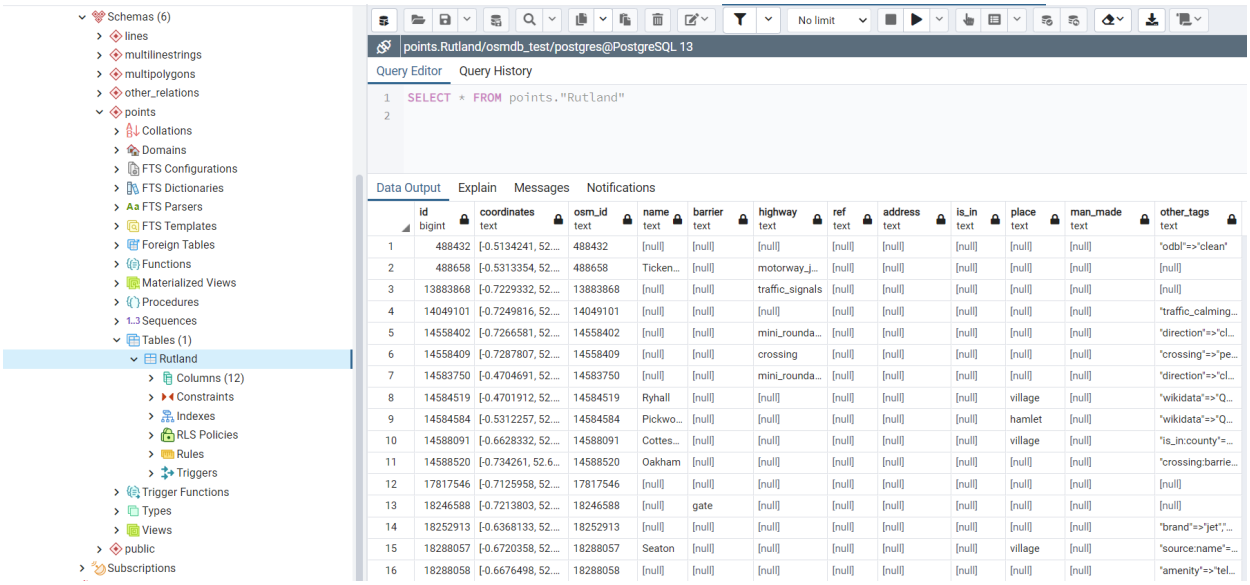

Fig. 8: An illustration of table name for storing the *'points'* layer of the OSM PBF data of Rutland.

2.3.2 Fetch data from the database

To fetch all the imported PBF data of Rutland, you can use the method *.fetch_osm_data()*:

```
>>> rutland_pbf_parsed_ = osmdb_test.fetch_osm_data(subregion_name, layer_names=None,
...                                                 decode_wkt=True)
```

You could find that `rutland_pbf_parsed_` is an equivalent of `rutland_pbf_parsed`:

```
>>> check_equivalence = all(
...     rutland_pbf_parsed[lyr_name].equals(rutland_pbf_parsed_[lyr_name])
...     for lyr_name in rutland_pbf_parsed_.keys())

>>> print(f"`rutland_pbf_parsed_` equals `rutland_pbf_parsed`: {check_equivalence}")
`rutland_pbf_parsed_` equals `rutland_pbf_parsed`: True
```

Note:

- The parameter `layer_names` is `None` by default, meaning that you're going to fetch data of all layers available from the database.
- The data stored in the database was parsed by the method *.read_osm_pbf()* given `parse_raw_feat=True` (see *above*). When it is being imported in the PostgreSQL server, the data type of the column 'coordinates' is converted from list to str. Therefore, in the above example of using the method *.fetch_osm_data()*, the parameter `decode_wkt` was set to `True` to retrieve the same data.

2.3.3 Import/fetch specific layers of shapefile

Of course, you can also import/fetch data of only a specific layer or multiple layers (and in a customised order). For example, let's firstly import the transport-related layers of Birmingham shapefile data.

Note: 'Birmingham' is not listed on the free download catalogue of Geofabrik, but that of BBBike. You need to change the data source to 'BBBike' for the instance `osmdb_test` (see also the *note* above).

```
>>> osmdb_test.DataSource = 'BBBike'

>>> subregion_name = 'Birmingham'

>>> birmingham_shp = osmdb_test.Reader.read_shp_zip(subregion_name, data_dir=data_dir,
...                                                 verbose=True)
To download .shp.zip data of the following geographic region(s):
    Birmingham
? [No]|Yes: yes
Downloading "Birmingham.osm.shp.zip" to "tests\" ... Done.
Extracting "tests\Birmingham.osm.shp.zip" ...
to "tests\"
Done.
Parsing files at "tests\Birmingham-shp\shape\" ... Done.

>>> type(birmingham_shp)
dict

>>> # Check names of layers included in the data
>>> list(birmingham_shp.keys())
['buildings',
 'landuse',
 'natural',
 'places',
 'points',
 'railways',
 'roads',
 'waterways']

>>> # Import the data of 'railways', 'roads' and 'waterways'
>>> lyr_names = ['railways', 'roads', 'waterways']

>>> osmdb_test.import_osm_data(birmingham_shp, subregion_name, lyr_names, verbose=True)
To import data into table "Birmingham" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
    "railways" ... Done: <total of rows> features.
    "roads" ... Done: <total of rows> features.
    "waterways" ... Done: <total of rows> features.
```

As illustrated in Fig. 9, three schemas: 'railways', 'roads' and 'waterways' are created in the *'osmdb_test'* database for storing the data of the three shapefile layers of Birmingham.

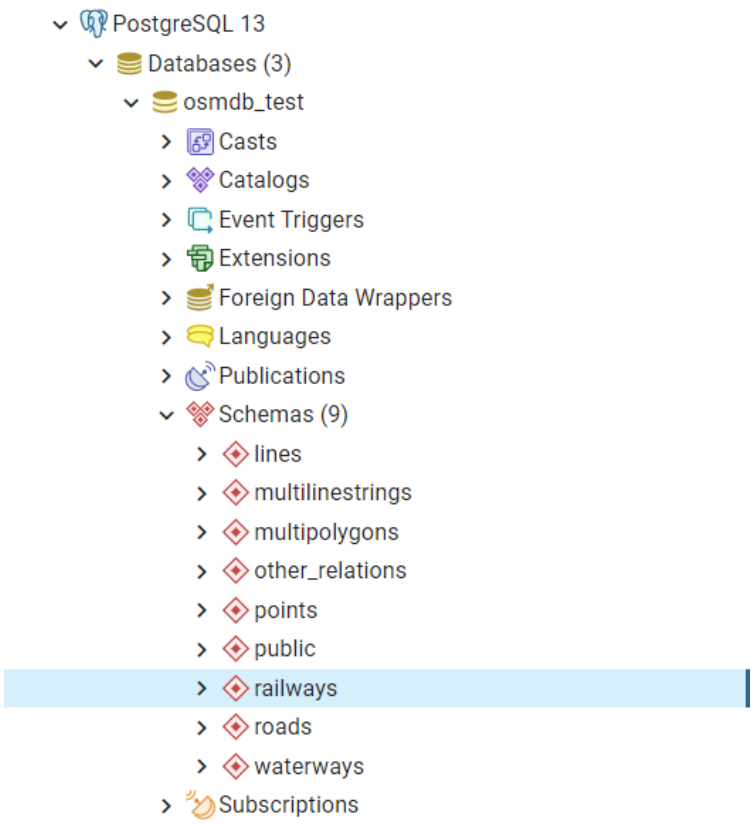

Fig. 9: An illustration of the newly created schemas for the selected layers of Birmingham shapefile data.

To fetch only the 'railways' data of Birmingham:

```
>>> lyr_name = 'railways'

>>> birmingham_shp_ = osmdb_test.fetch_osm_data(subregion_name, layer_names=lyr_name,
...                                             decode_wkt=True, sort_by='osm_id')

>>> # This is dict-type
>>> type(birmingham_shp_)
dict

>>> # Data frame of the 'railways' layer
>>> birmingham_shp_railways_ = birmingham_shp_[lyr_name]

>>> birmingham_shp_railways_.head()
    osm_id  ... shape_type
0      740  ...          3
1     2148  ...          3
2  2950000  ...          3
3  3491845  ...          3
4  3981454  ...          3
[5 rows x 5 columns]

>>> birmingham_shp_railways_.columns.tolist()
['osm_id', 'name', 'type', 'coordinates', 'shape_type']
```

Note: The data retrieved from a PostgreSQL database may not be in the same order as it is in the database (see the test code below). However, they contain exactly the same information.

You may sort the data by `id` (or `osm_id`) to make a comparison.

```
>>> birmingham_shp_railways = birmingham_shp[lyr_name]

>>> birmingham_shp_railways.head()
    osm_id  ... shape_type
0      740  ...          3
1     2148  ...          3
2  2950000  ...          3
3  3491845  ...          3
4  3981454  ...          3
[5 rows x 5 columns]

>>> birmingham_shp_railways.columns.tolist()
['osm_id', 'name', 'type', 'coordinates', 'shape_type']
```

Note:

- `birmingham_shp_railways` and `birmingham_shp_railways_` both pandas.DataFrame.
- It must be noted that empty strings, `''`, are automatically saved as `None` when importing `birmingham_shp` into the PostgreSQL database. Therefore, the retrieved `birmingham_shp_railways_` may not be exactly equal to `birmingham_shp_railways`.

```
>>> check_eq = birmingham_shp_railways_.equals(birmingham_shp_railways)

>>> print(f"`birmingham_shp_railways_` equals `birmingham_shp_railways`: {check_eq}")
`birmingham_shp_railways_` equals `birmingham_shp_railways`: False

>>> # Try filling ``None`` values with ``''``
>>> birmingham_shp_railways_.fillna('', inplace=True)

>>> # Check again the equivalence
>>> check_eq = birmingham_shp_railways_.equals(birmingham_shp_railways)
>>> print(f"`birmingham_shp_railways_` equals `birmingham_shp_railways`: {check_eq}")
`birmingham_shp_railways_` equals `birmingham_shp_railways`: True
```

2.3.4 Drop data

If you would now like to drop the data of all or selected layers that have been imported for one or multiple geographic regions, you can use the method *`.drop_subregion_table()`*.

For example, to drop the 'railways' data of Birmingham:

```
>>> # Recall that: subregion_name == 'Birmingham'; lyr_name == 'railways'

>>> osmdb_test.drop_subregion_table(subregion_name, lyr_name, verbose=True)
To drop table "railways"."Birmingham" from postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Dropping the table ...
    "railways"."Birmingham" ... Done.
```

To also drop the 'waterways' of Birmingham and both 'lines' and 'multilinestrings' of Rutland:

```
>>> subregion_names = ['Birmingham', 'Rutland']
>>> lyr_names = ['waterways', 'lines', 'multilinestrings']

>>> osmdb_test.drop_subregion_table(subregion_names, lyr_names, verbose=True)
To drop tables from postgres:***@localhost:5432/osmdb_test:
    "Birmingham"
    "Rutland"
 under the schemas:
    "lines"
    "multilinestrings"
    "waterways"
? [No]|Yes: yes
Dropping the tables ...
    "lines"."Rutland" ... Done.
    "multilinestrings"."Rutland" ... Done.
    "waterways"."Birmingham" ... Done.
```

You could also easily drop the whole database 'osmdb_test' if you don't need it anymore:

```
>>> osmdb_test.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

2.4 Clear up 'the mess' in here

To remove all the data files that have been downloaded and generated:

```
>>> from pyhelpers.dir import cd, delete_dir

>>> list_of_data_dirs = ['Birmingham-shp', 'greater-london_kent_railways']

>>> for dat_dir in list_of_data_dirs:
...     delete_dir(cd(data_dir, dat_dir), confirmation_required=False, verbose=True)
Deleting "tests\Birmingham-shp\" ... Done.
Deleting "tests\greater-london_kent_railways\" ... Done.

>>> list_of_data_files = ['Birmingham.osm.shp.zip',
...                       'greater-london-latest.osm.pbf',
...                       'greater-london-latest-free.shp.zip',
...                       'kent-latest-free.shp.zip',
...                       'rutland-latest.osm.pbf',
...                       'west-midlands-latest.osm.pbf',
...                       'west-yorkshire-latest.osm.pbf']

>>> for dat_file in list_of_data_files:
...     rel_file_path = os.path.relpath(cd(data_dir, dat_file))
...     print("Deleting \"{}\"".format(rel_file_path), end=" ... ")
...     try:
...         os.remove(rel_file_path)
...         print("Done.")
...     except Exception as e:
...         print("Failed. {}".format(e))
Deleting "tests\Birmingham.osm.shp.zip" ... Done.
Deleting "tests\greater-london-latest.osm.pbf" ... Done.
Deleting "tests\greater-london-latest-free.shp.zip" ... Done.
Deleting "tests\kent-latest-free.shp.zip" ... Done.
Deleting "tests\rutland-latest.osm.pbf" ... Done.
```

(continues on next page)

(continued from previous page)

```
Deleting "tests\west-midlands-latest.osm.pbf" ... Done.
Deleting "tests\west-yorkshire-latest.osm.pbf" ... Done.

>>> # # To remove the "tests" directory
>>> # delete_dir(cd(data_dir))
```

(**THE END of** *Quick start*.)

For more details, check out *Modules*.

CHAPTER

THREE

MODULES

The package includes the following six modules:

downloader	Download OpenStreetMap (OSM) data extracts from the free download servers of Geofabrik and BBBike.
reader	Read OpenStreetMap (OSM) data extracts.
ios	I/O and storage of OSM data extracts with PostgreSQL.
utils	Helper functions.
settings	Default settings for working environment.
updater	Updating package data.

3.1 downloader

Download OpenStreetMap (OSM) data extracts from the free download servers of Geofabrik and BBBike.

Classes

GeofabrikDownloader([download_dir])	Download OSM data from Geofabrik free download server.
BBBikeDownloader([download_dir])	Download OSM data from BBBike free download server.

3.1.1 GeofabrikDownloader

class pydriosm.downloader.**GeofabrikDownloader**(*download_dir=None*)

Download OSM data from Geofabrik free download server.

Parameters **download_dir** – (a path or a name of) a directory for saving downloaded data files; if None (default), a folder osm_geofabrik under the current working directory

Variables

- ***Name*** (*str*) – name of data
- **Abbr** (*str*) – short name of the data
- ***URL*** (*str*) – URL of the homepage to the free download server
- **DownloadIndexURL** (*str*) – URL of the official download index
- **ValidFileFormats** (*list*) – valid file formats available on the free download server

Example:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> print(geofabrik_downloader.Name)
Geofabrik OpenStreetMap data extracts

>>> print(geofabrik_downloader.URL)
https://download.geofabrik.de/
```

Methods

download_osm_data(subregion_names, ...[, ...])	Download OSM data (in a specific format) of one (or multiple) geographic region(s).
download_subregion_data(subregion_nar ...)	Download OSM data (in a specific file format) of all subregions (if available) for one (or multiple) geographic region(s).
file_exists(subregion_name, osm_file_format)	Check if a requested data file of a geographic region already exists locally, given its default filename.
get_continents_subregion_tables([upc ...])	Get download information for continents.
get_default_osm_filename(subregion_na ...)	get a default filename for a geograpic region.
get_default_path_to_osm_file(subregic ...)	Get a default path to a local directory for storing a downloaded data file.
get_download_catalogue([update, ...])	Get a catalogue of download information.
get_download_index([update, ...])	Get the formal index of all available downloads.

continues on next page

Table 3 – continued from previous page	
get_list_of_subregion_names([update, ...])	Get a list of names of all available geographic regions.
get_raw_directory_index(url[, verbose])	Get a raw directory index.
get_region_subregion_tier([update, ...])	Get a catalogue of region-subregion tier.
get_subregion_download_url(subregion_ ...)	Get a download URL of a geographic region.
get_subregion_table(url[, verbose])	Get download information for all geographic regions on a web page.
make_sub_download_dir(subregion_name, ...[, ...])	Make a default directory for downloading data of a geographic region's subregions.
search_for_subregions(*subregion_name deep])	Retrieve names of all subregions (if any) of the given geographic region(s).
validate_input_file_format(osm_file_f	Validate an input file format of OSM data.
validate_input_subregion_name(subreg	Validate an input name of a geographic region.

GeofabrikDownloader.download_osm_data

GeofabrikDownloader.**download_osm_data**(*subregion_names*, *osm_file_format*, *download_dir=None*, *update=False*, *confirmation_required=True*, *deep_retry=False*, *interval=None*, *verbose=False*, *ret_download_path=False*, ***kwargs*)

Download OSM data (in a specific format) of one (or multiple) geographic region(s).

Parameters

- **subregion_names** (*str or list*) – name of a geographic region (or names of multiple geographic regions) available on Geofabrik free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **download_dir** (*str or None*) – directory for saving the downloaded file(s); if None (default), the default directory created by the package
- **update** (*bool*) – whether to update the data if it already exists, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **deep_retry** (*bool*) – whether to further check availability of sub-subregions data, defaults to False
- **interval** (*int or None*) – interval (in sec) between downloading two subregions, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

- **ret_download_path** (*bool*) – whether to return the path(s) to the downloaded file(s), defaults to `False`
- **kwargs** – optional parameters of pyhelpers.ops.download_file_from_url()

Returns absolute path(s) to downloaded file(s) when `ret_download_path` is `True`

Return type list or str

Examples:

```
>>> import os
>>> from pyhelpers.dir import delete_dir
>>> from pydriosm.downloader import GeofabrikDownloader, cd_dat_geofabrik

>>> geofabrik_downloader = GeofabrikDownloader()

>>> # Download PBF data file of Greater London and Rutland
>>> region_names = ['London', 'Rutland']  # Case-insensitive
>>> file_format = ".pbf"

>>> dwnld_paths = geofabrik_downloader.download_osm_data(region_names, file_format,
...                                                      verbose=True,
...                                                      ret_download_path=True)
To download .osm.pbf data of the following geographic region(s):
    Greater London
    Rutland
? [No]|Yes: yes
Downloading "greater-london-latest.osm.pbf" to "... ...\Greater London\" ... Done.
Downloading "rutland-latest.osm.pbf" to "... ...\Rutland\" ... Done.

>>> for dwnld_path in dwnld_paths:
...     print(os.path.relpath(dwnld_path))
osm_geofabrik\Europe\Great Britain\England\Greater London\greater-lond...
osm_geofabrik\Europe\Great Britain\England\Rutland\rutland-latest.osm.pbf

>>> # Delete the directory generated above
>>> delete_dir(cd_dat_geofabrik(), verbose=True)
The directory "osm_geofabrik\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "osm_geofabrik\" ... Done.

>>> # Download shapefiles of West Midlands
>>> region_name = 'west midlands'  # Case-insensitive
>>> file_format = ".shp"
>>> dwnld_dir = "tests"

>>> dwnld_path = geofabrik_downloader.download_osm_data(region_name, file_format,
...                                                     dwnld_dir, verbose=True,
...                                                     ret_download_path=True)
To download .shp.zip data of the following geographic region(s):
    West Midlands
? [No]|Yes: yes
Downloading "west-midlands-latest-free.shp.zip" to "tests\" ... Done.

>>> print(os.path.relpath(dwnld_path))
tests\west-midlands-latest-free.shp.zip

>>> # Delete the downloaded .shp.zip file
>>> os.remove(dwnld_path)
```

(continues on next page)

(continued from previous page)

```
>>> # Download shapefiles of Great Britain
>>> region_name = 'Great Britain'  # Case-insensitive
>>> file_format = ".shp"

>>> # By default, `deep_retry` is `False`
>>> dwnld_path = geofabrik_downloader.download_osm_data(region_name, file_format,
...                                                     dwnld_dir, verbose=True,
...                                                     ret_download_path=True)
To download .shp.zip data of the following geographic region(s):
    Great Britain
? [No]|Yes: yes
No .shp.zip data is found for "Great Britain".
Try to download the data of its subregions instead
? [No]|Yes: yes
Downloading "england-latest-free.shp.zip" to "tests\great-britain-sh...\ ... Done.
Downloading "scotland-latest-free.shp.zip" to "tests\great-britain-s...\ ... Done.
Downloading "wales-latest-free.shp.zip" to "tests\great-britain-shp-zip\ ... Done.

>>> # Set `deep_retry` to `True`
>>> dwnld_path = geofabrik_downloader.download_osm_data(region_name, file_format,
...                                                     dwnld_dir, deep_retry=True,
...                                                     verbose=True,
...                                                     ret_download_path=True)
To download .shp.zip data of the following geographic region(s):
    Great Britain
? [No]|Yes: yes
No .shp.zip data is found for "Great Britain".
Try to download the data of its subregions instead
? [No]|Yes: yes
"scotland-latest-free.shp.zip" is already available at "tests\great-britain-...\".
"wales-latest-free.shp.zip" is already available at "tests\great-britain-shp...\".
Downloading "bedfordshire-latest-free.shp.zip" to "tests\great-britain-\ ... Done.
... ...
Downloading "west-yorkshire-latest-free.shp.zip" to "tests\great-bri...\ ... Done.
Downloading "wiltshire-latest-free.shp.zip" to "tests\great-britain-...\ ... Done.
Downloading "worcestershire-latest-free.shp.zip" to "tests\great-bri...\ ... Done.

>>> # Check the file paths
>>> len(dwnld_path)
49
>>> os.path.commonpath(dwnld_path) == geofabrik_downloader.DownloadDir
True
>>> print(os.path.relpath(geofabrik_downloader.DownloadDir))
tests\great-britain-shp-zip

>>> # Delete the downloaded files
>>> delete_dir(geofabrik_downloader.DownloadDir, verbose=True)
The directory "tests\great-britain-shp-zip\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "tests\great-britain-shp-zip\" ... Done.
```

GeofabrikDownloader.download_subregion_data

GeofabrikDownloader.**download_subregion_data**(*subregion_names*, *osm_file_format*, *download_dir=None*, *deep=False*, *update=False*, *confirmation_required=True*, *interval=None*, *verbose=False*, *ret_download_path=False*, ***kwargs*)

Download OSM data (in a specific file format) of all subregions (if available) for one (or multiple) geographic region(s).

If no subregion data is available for the region(s) specified by `subregion_names`, then the data of `subregion_names` would be downloaded only.

Parameters

- **subregion_names** (*str or list*) – name of a geographic region (or names of multiple geographic regions) available on Geofabrik free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **download_dir** (*str or None*) – directory for saving the downloaded file(s); if `None` (default), the default directory created by the package
- **deep** (*bool*) – whether to try to search for subregions of subregion(s), defaults to `False`
- **update** (*bool*) – whether to update the data if it already exists, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **interval** (*int or None*) – interval (in second) between downloading two subregions, defaults to `None`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **ret_download_path** (*bool*) – whether to return the path(s) to the downloaded file(s), defaults to `False`
- **kwargs** – optional parameters of pydriosm.GeofabrikDownloader.download_osm_data()

Returns the path(s) to the downloaded file(s) when `ret_download_path` is `True`

Return type list or str

Examples:

```
>>> import os
>>> from pyhelpers.dir import cd
>>> from pydriosm.downloader import GeofabrikDownloader
```

(continues on next page)

(continued from previous page)

```
>>> geofabrik_downloader = GeofabrikDownloader()

>>> region_names = ['rutland', 'west yorkshire']
>>> file_format = ".pbf"
>>> dwnld_dir = "tests"

>>> geofabrik_downloader.download_subregion_data(
...     region_names, file_format, dwnld_dir, verbose=True)
To download .osm.pbf data of the following geographic region(s):
    Rutland
    West Yorkshire
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "tests\" ... Done.
Downloading "west-yorkshire-latest.osm.pbf" to "tests\" ... Done.

>>> # Delete "tests\rutland-latest.osm.pbf"
>>> os.remove(cd(dwnld_dir, "rutland-latest.osm.pbf"))

>>> # Try to download data given another list which also includes 'West Yorkshire'
>>> region_names = ['west midlands', 'west yorkshire']

>>> dwnld_paths = geofabrik_downloader.download_subregion_data(
...     region_names, file_format, dwnld_dir, verbose=True, ret_download_path=True)
"west-yorkshire-latest.osm.pbf" is already available at "tests\".
To download .osm.pbf data of the following geographic region(s):
    West Midlands
? [No]|Yes: yes
Downloading "west-midlands-latest.osm.pbf" to "tests\" ... Done.

>>> for dwnld_path in dwnld_paths:
...     print(os.path.relpath(dwnld_path))
tests\west-midlands-latest.osm.pbf
tests\west-yorkshire-latest.osm.pbf

>>> # Update (or re-download) the existing data file
>>> dwnld_paths = geofabrik_downloader.download_subregion_data(
...     region_names, file_format, dwnld_dir, update=True, verbose=True,
...     ret_download_path=True)
"west-midlands-latest.osm.pbf" is already available at "tests\".
"west-yorkshire-latest.osm.pbf" is already available at "tests\".
To update the .osm.pbf data of the following geographic region(s):
    West Midlands
    West Yorkshire
? [No]|Yes: yes
Updating "west-midlands-latest.osm.pbf" at "tests\" ... Done.
Updating "west-yorkshire-latest.osm.pbf" at "tests\" ... Done.

>>> # To download the PBF data of England
>>> region_names = 'England'

>>> dwnld_paths = geofabrik_downloader.download_subregion_data(
...     region_names, file_format, dwnld_dir, update=True, verbose=True,
...     ret_download_path=True)
"west-midlands-latest.osm.pbf" is already available at "tests\".
"west-yorkshire-latest.osm.pbf" is already available at "tests\".
To download/update the .osm.pbf data of the following geographic region(s):
    Bedfordshire
    Berkshire
    ...
```

(continues on next page)

(continued from previous page)

```
    West Midlands
    ...
    West Yorkshire
    Wiltshire
    Worcestershire
? [No]|Yes: yes
Downloading "bedfordshire-latest.osm.pbf" to "tests\" ... Done.
Downloading "berkshire-latest.osm.pbf" to "tests\" ... Done.
...
Updating "west-midlands-latest.osm.pbf" at "tests\" ... Done.
...
Updating "west-yorkshire-latest.osm.pbf" at "tests\" ... Done.
Downloading "wiltshire-latest.osm.pbf" to "tests\" ... Done.
Downloading "worcestershire-latest.osm.pbf" to "tests\" ... Done.

>>> len(dwnld_paths)
47

>>> # Delete the downloaded files
>>> for dwnld_path in dwnld_paths:
...     os.remove(dwnld_path)
```

GeofabrikDownloader.file_exists

GeofabrikDownloader.**file_exists**(*subregion_name*, *osm_file_format*, *data_dir=None*, *update=False*, *verbose=False*, *ret_file_path=False*)

Check if a requested data file of a geographic region already exists locally, given its default filename.

Parameters

- **subregion_name** (*str*) – name of a geographic region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **data_dir** (*str or None*) – directory for saving the downloaded file(s); if None (default), the default directory created by the package
- **update** (*bool*) – whether to (check on and) update the data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **ret_file_path** (*bool*) – whether to return the path to the data file (if it exists), defaults to False

Returns whether or not the requested data file exists; or the path to the data file

Return type bool or str

Examples:

```
>>> import os
>>> from pyhelpers.dir import delete_dir
>>> from pydriosm.downloader import GeofabrikDownloader, cd_dat_geofabrik

>>> geofabrik_downloader = GeofabrikDownloader()

>>> region_name = 'london'
>>> file_format = ".pbf"

>>> # Download the PBF data of London (to the default directory)
>>> geofabrik_downloader.download_osm_data(region_name, file_format, verbose=True)
To download .osm.pbf data of the following geographic region(s):
    Greater London
? [No]|Yes: yes
Downloading "greater-london-latest.osm.pbf" to "downloads_G\...\England" ... Done.

>>> # Check whether the PBF data file exists; `ret_file_path` is by default `False`
>>> pbf_exists = geofabrik_downloader.file_exists(region_name, file_format)

>>> type(pbf_exists)
bool
>>> # If the data file exists at the default directory created by the package
>>> print(pbf_exists)
True

>>> # Set `ret_file_path` to be `True`
>>> path_to_pbf = geofabrik_downloader.file_exists(
...     subregion_name=region_name, osm_file_format=file_format, ret_file_
 ↪path=True)

>>> # If the data file exists at the default directory created by the package:
>>> type(path_to_pbf)
str
>>> print(os.path.relpath(path_to_pbf))
osm_geofabrik\Europe\Great Britain\England\greater-london-latest.osm.pbf

>>> # Remove the directory or the PBF file and check again:
>>> delete_dir(cd_dat_geofabrik(), confirmation_required=False, verbose=True)
Deleting "osm_geofabrik" ... Done.
>>> path_to_pbf = geofabrik_downloader.file_exists(
...     subregion_name=region_name, osm_file_format=file_format, ret_file_
 ↪path=True)

>>> # Since the data file does not exist at the default directory
>>> type(path_to_pbf)
bool
>>> print(path_to_pbf)
False
```

GeofabrikDownloader.get_continents_subregion_tables

GeofabrikDownloader.**get_continents_subregion_tables**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get download information for continents.

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to False

- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns subregion information for each continent

Return type dict or None

Example:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> # Information of subregions for each continent
>>> subregion_tbls = geofabrik_downloader.get_continents_subregion_tables()

>>> type(subregion_tbls)
dict

>>> list(subregion_tbls.keys())
['Africa',
 'Antarctica',
 'Asia',
 'Australia and Oceania',
 'Central America',
 'Europe',
 'North America',
 'South America']

>>> # Information about the data of subregions in Asia
>>> asia_tbl = subregion_tbls['Asia']

>>> type(asia_tbl)
pandas.core.frame.DataFrame

>>> asia_tbl.head()
     Subregion  ...                                            .osm.bz2
0  Afghanistan  ...  https://download.geofabrik.de/asia/afghanistan...
1      Armenia  ...  https://download.geofabrik.de/asia/armenia-lat...
2   Azerbaijan  ...  https://download.geofabrik.de/asia/azerbaijan-...
3   Bangladesh  ...  https://download.geofabrik.de/asia/bangladesh-...
4       Bhutan  ...  https://download.geofabrik.de/asia/bhutan-late...
[5 rows x 6 columns]

>>> asia_tbl.columns.tolist()
['Subregion',
 'SubregionURL',
 '.osm.pbf',
 '.osm.pbf.Size',
 '.shp.zip',
 '.osm.bz2']
```

GeofabrikDownloader.get_default_osm_filename

GeofabrikDownloader.**get_default_osm_filename**(*subregion_name*, *osm_file_format*, *update=False*)

get a default filename for a geograpic region.

The default filename is derived from the relevant download URL for the requested data file.

Parameters

- **subregion_name** (*str*) – name of a geographic region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **update** (*bool*) – whether to (check on and) update the package data, defaults to False

Returns default OSM filename for the subregion_name

Return type str or None

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> geo_region_name = 'london'
>>> file_format = ".pbf"

>>> # Default filename of the PBF data of London
>>> fn = geofabrik_downloader.get_default_osm_filename(geo_region_name, file_
 ↪format)

>>> print(fn)
greater-london-latest.osm.pbf

>>> geo_region_name = 'britain'
>>> file_format = ".shp"

>>> # Default filename of the shapefile data of Great Britain
>>> fn = geofabrik_downloader.get_default_osm_filename(geo_region_name, file_
 ↪format)
No .shp.zip data for Great Britain is available to download.

>>> print(fn)
None
```

GeofabrikDownloader.get_default_path_to_osm_file

GeofabrikDownloader.**get_default_path_to_osm_file**(*subregion_name*, *osm_file_format*, *mkdir=False*, *update=False*, *verbose=False*)

Get a default path to a local directory for storing a downloaded data file.

The default file path is derived from the relevant download URL for the requested data file.

Parameters

- **subregion_name** (*str*) – name of a geographic region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **update** (*bool*) – whether to (check on and) update the package data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns default filename of the subregion and default (absolute) path to the file

Return type tuple

Example:

```
>>> import os
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> # Default filename and download path of the PBF data of London
>>> filename, file_path = geofabrik_downloader.get_default_path_to_osm_file(
...     subregion_name='London', osm_file_format=".pbf")

>>> print(filename)
greater-london-latest.osm.pbf

>>> print(os.path.relpath(file_path))
osm_geofabrik\Europe\Great Britain\England\Greater London\greater-lond...
```

GeofabrikDownloader.get_download_catalogue

GeofabrikDownloader.**get_download_catalogue**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get a catalogue of download information.

Similar to the method *.get_download_index()*.

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns a catalogue for all subregion downloads

Return type pandas.DataFrame or None

Example:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> # A download catalogue for all subregions
>>> downloads_catalogue = geofabrik_downloader.get_download_catalogue()

>>> type(downloads_catalogue)
pandas.core.frame.DataFrame

>>> downloads_catalogue.head()
      Subregion  ...                                          .osm.bz2
0       Algeria  ...  https://download.geofabrik.de/africa/algeria-l...
1        Angola  ...  https://download.geofabrik.de/africa/angola-la...
2         Benin  ...  https://download.geofabrik.de/africa/benin-lat...
3      Botswana  ...  https://download.geofabrik.de/africa/botswana-...
4  Burkina Faso  ...  https://download.geofabrik.de/africa/burkina-f...
[5 rows x 6 columns]

>>> downloads_catalogue.columns.tolist()
['Subregion',
 'SubregionURL',
 '.osm.pbf',
 '.osm.pbf.Size',
 '.shp.zip',
 '.osm.bz2']
```

GeofabrikDownloader.get_download_index

GeofabrikDownloader.**get_download_index**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get the formal index of all available downloads.

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns the formal index of all downloads

Return type pandas.DataFrame or None

Example:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> # The formal index of all available downloads
>>> download_idx = geofabrik_downloader.get_download_index()

>>> type(download_idx)
pandas.core.frame.DataFrame

>>> download_idx.columns.tolist()
['id',
 'parent',
 'name',
 'urls',
 'geometry',
 'pbf',
 'bz2',
 'shp',
 'pbf-internal',
 'history',
 'taginfo',
 'updates']

>>> download_idx.head()
           id  ...                                            updates
0  afghanistan  ...  https://download.geofabrik.de/asia/afghanistan...
1       africa  ...       https://download.geofabrik.de/africa-updates
2      albania  ...  https://download.geofabrik.de/europe/albania-u...
3      alberta  ...  https://download.geofabrik.de/north-america/ca...
4      algeria  ...  https://download.geofabrik.de/africa/algeria-u...
[5 rows x 12 columns]
```

GeofabrikDownloader.get_list_of_subregion_names

GeofabrikDownloader.**get_list_of_subregion_names**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get a list of names of all available geographic regions.

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns names of geographic regions available on the free download server

Return type list

Example:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> # A list of the names of available geographic regions
>>> geo_region_name_list = geofabrik_downloader.get_list_of_subregion_names()

>>> geo_region_name_list[:5]
['Algeria', 'Angola', 'Benin', 'Botswana', 'Burkina Faso']
```

GeofabrikDownloader.get_raw_directory_index

static GeofabrikDownloader.**get_raw_directory_index**(*url*, *verbose=False*)

Get a raw directory index.

This includes logs of older files and their and download URLs.

Parameters

- **url** (*str*) – URL to the web page of the homepage or any subregion
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns data of raw directory index

Return type pandas.DataFrame or None

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> gb_url = 'https://download.geofabrik.de/europe/great-britain.html'
```

(continues on next page)

(continued from previous page)

```
>>> raw_dir_idx = geofabrik_downloader.get_raw_directory_index(gb_url)

>>> type(raw_dir_idx)
pandas.core.frame.DataFrame
>>> raw_dir_idx.head()
                              File  ...                                    ␣
 ↪FileURL
0             great-britain-updates  ...  https://download.geofabrik.de/europe/gr..
 ↪.
1  great-britain-210412.osm.pbf.md5  ...  https://download.geofabrik.de/europe/gr..
 ↪.
2  great-britain-latest.osm.pbf.md5  ...  https://download.geofabrik.de/europe/gr..
 ↪.
3                 great-britain.kml  ...  https://download.geofabrik.de/europe/gr..
 ↪.
4      great-britain-latest.osm.pbf  ...  https://download.geofabrik.de/europe/gr..
 ↪.
[5 rows x 4 columns]

>>> gf_url = 'https://download.geofabrik.de/'

>>> raw_dir_idx = geofabrik_downloader.get_raw_directory_index(gf_url,␣
 ↪verbose=True)
Collecting the raw directory index for the page 'https://download.ge...' ...␣
 ↪Failed.
The web page does not have any raw directory index.
>>> type(raw_dir_idx)
NoneType
```

GeofabrikDownloader.get_region_subregion_tier

GeofabrikDownloader.**get_region_subregion_tier**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get a catalogue of region-subregion tier.

This includes all geographic regions to which data of subregions is unavailable.

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns region-subregion tier (in `dict` type) and all that have no subregions (in `list` type)

Return type tuple

Example:

```
>>> from pydriosm.downloader import GeofabrikDownloader
```

(continues on next page)

(continued from previous page)

```
>>> geofabrik_downloader = GeofabrikDownloader()

>>> # region-subregion tier, and all regions that have no subregions
>>> rs_tier, ns_list = geofabrik_downloader.get_region_subregion_tier()

>>> # Keys of the region-subregion tier
>>> list(rs_tier.keys())
['Africa',
 'Antarctica',
 'Asia',
 'Australia and Oceania',
 'Central America',
 'Europe',
 'North America',
 'South America']

>>> # A sample of five regions that have no subregions
>>> ns_list[0:5]
['Antarctica', 'Algeria', 'Angola', 'Benin', 'Botswana']
```

GeofabrikDownloader.get_subregion_download_url

GeofabrikDownloader.**get_subregion_download_url**(*subregion_name*, *osm_file_format*, *update=False*, *verbose=False*)

Get a download URL of a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **update** (*bool*) – whether to (check on and) update the package data, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns name and URL of the subregion

Return type tuple

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> geo_region_name = 'England'
>>> osm_file_fmt = ".pbf"

>>> fml_name, dwnld_link = geofabrik_downloader.get_subregion_download_url(
...     subregion_name=geo_region_name, osm_file_format=osm_file_fmt)
```

(continues on next page)

(continued from previous page)

```
>>> print(fml_name)  # The name of the subregion on the free downloader server
England
>>> print(dwnld_link)  # The URL to the PBF data file
https://download.geofabrik.de/europe/great-britain/england-latest.osm.pbf

>>> geo_region_name = 'Britain'
>>> osm_file_fmt = ".shp"

>>> fml_name, dwnld_link = geofabrik_downloader.get_subregion_download_url(
...     subregion_name=geo_region_name, osm_file_format=osm_file_fmt)

>>> print(fml_name)
Great Britain
>>> print(dwnld_link)
None
```

GeofabrikDownloader.get_subregion_table

GeofabrikDownloader.**get_subregion_table**(*url*, *verbose=False*)

Get download information for all geographic regions on a web page.

Parameters

- **url** (*str*) – URL to the web resource
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns a table of all available subregions' URLs

Return type pandas.DataFrame or None

Example:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> gb_url = 'https://download.geofabrik.de/europe/great-britain.html'

>>> subregion_tbl = geofabrik_downloader.get_subregion_table(gb_url)

>>> type(subregion_tbl)
pandas.core.frame.DataFrame

>>> subregion_tbl.columns.tolist()
['Subregion',
 'SubregionURL',
 '.osm.pbf',
 '.osm.pbf.Size',
 '.shp.zip',
 '.osm.bz2']

>>> subregion_tbl.head()
  Subregion  ...                                          .osm.bz2
0   England  ...  https://download.geofabrik.de/europe/great-bri...
1  Scotland  ...  https://download.geofabrik.de/europe/great-bri...
2     Wales  ...  https://download.geofabrik.de/europe/great-bri...
[3 rows x 6 columns]
```

(continues on next page)

(continued from previous page)

```
>>> a_url = 'https://download.geofabrik.de/antarctica.html'

>>> subregion_tbl = geofabrik_downloader.get_subregion_table(a_url, verbose=True)
Collecting download information for "Antarctica" ... Checked out.
No more subregion data is available on the page 'https://download.geofabrik.de/...
 ↪'.

>>> type(subregion_tbl)
NoneType
```

GeofabrikDownloader.make_sub_download_dir

GeofabrikDownloader.**make_sub_download_dir**(*subregion_name*, *osm_file_format*, *download_dir=None*, *mkdir=False*)

Make a default directory for downloading data of a geographic region's subregions.

This is particularly useful when data of a geographic region and requested file format is unavailable.

Parameters

- **subregion_name** (*str*) – name of a geographic region available on Geofabrik free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **download_dir** (*str or None*) – directory for saving the downloaded file(s); if None (default), the default directory created by the package
- **mkdir** (*bool*) – whether to create a directory, defaults to False

Returns default download directory if the requested data file is not available

Return type str

Example:

```
>>> import os
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> region_name = 'london'
>>> file_format = ".pbf"

>>> # Default download directory (if the requested data file is not available)
>>> dwnld_dir = geofabrik_downloader.make_sub_download_dir(region_name, file_
 ↪format)

>>> print(os.path.relpath(dwnld_dir))
osm_geofabrik\Europe\Great Britain\England\Greater London\greater-lond...

>>> region_name = 'britain'
>>> file_format = ".shp"
```

(continues on next page)

(continued from previous page)

```
>>> # Default download directory (if the requested data file is not available)
>>> dwnld_dir = geofabrik_downloader.make_sub_download_dir(region_name, file_
 ↪format,
...                                                       download_dir="tests")

>>> print(os.path.relpath(dwnld_dir))
tests\Great Britain\great-britain-shp-zip
```

GeofabrikDownloader.search_for_subregions

GeofabrikDownloader.**search_for_subregions**(**subregion_name*, *deep=False*)

Retrieve names of all subregions (if any) of the given geographic region(s).

The is based on the region-subregion tier.

See also [RNS-1].

Parameters

- **subregion_name** (*str or None*) – name of a geographic region available on Geofabrik free download server
- **deep** (*bool*) – whether to get subregion names of the subregions, defaults to False

Returns list of subregions (if any); if subregion_name=None, all regions that do have subregions

Return type list

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> # Names of all (sub)regions
>>> region_names = geofabrik_downloader.search_for_subregions()

>>> len(region_names) > 400
True
>>> region_names[:5]
['Antarctica', 'Algeria', 'Angola', 'Benin', 'Botswana']
>>> region_names[-5:]
['centro-oeste', 'nordeste', 'norte', 'sudeste', 'sul']

>>> # Names of all subregions of England and North America
>>> region_names = geofabrik_downloader.search_for_subregions('england', 'n america
 ↪')

>>> len(region_names)
99
>>> region_names[:5]
['Bedfordshire', 'Berkshire', 'Bristol', 'Buckinghamshire', 'Cambridgeshire']
>>> region_names[-5:]
['Virginia', 'Washington', 'West Virginia', 'Wisconsin', 'Wyoming']

>>> # Names of subregions of Great Britain
>>> region_names = geofabrik_downloader.search_for_subregions('britain')
```

(continues on next page)

(continued from previous page)

```
>>> len(region_names)
3
>>> region_names
['England', 'Scotland', 'Wales']

>>> # Names of all subregions of Great Britain's subregions
>>> region_names = geofabrik_downloader.search_for_subregions('britain', deep=True)
>>> len(region_names)
49
>>> region_names[:5]
['Scotland', 'Wales', 'Bedfordshire', 'Berkshire', 'Bristol']
>>> region_names[-5:]
['West Midlands', 'West Sussex', 'West Yorkshire', 'Wiltshire', 'Worcestershire']
```

GeofabrikDownloader.validate_input_file_format

GeofabrikDownloader.**validate_input_file_format**(*osm_file_format*)

Validate an input file format of OSM data.

The validation is done by matching the input `osm_file_format` to a filename extension available on Geofabrik free download server.

Parameters **osm_file_format** (*str*) – filename extension of OSM data

Returns formal file format

Return type str

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> file_format = ".pbf"

>>> valid_file_format = geofabrik_downloader.validate_input_file_format(file_
 ↪format)
>>> print(valid_file_format)
.osm.pbf

>>> file_format = "shp"

>>> valid_file_format = geofabrik_downloader.validate_input_file_format(file_
 ↪format)
>>> print(valid_file_format)
.shp.zip
```

GeofabrikDownloader.validate_input_subregion_name

GeofabrikDownloader.**validate_input_subregion_name**(*subregion_name*)

Validate an input name of a geographic region.

The validation is done by matching the input `subregion_name` to a name of a geographic region available on Geofabrik free download server.

Parameters **subregion_name** (*str*) – name (or URL) of a geographic region

Returns valid subregion name that matches, or is the most similar to, the input `subregion_name`

Return type str

Examples:

```
>>> from pydriosm.downloader import GeofabrikDownloader

>>> geofabrik_downloader = GeofabrikDownloader()

>>> geo_region_name = 'london'

>>> valid_name = geofabrik_downloader.validate_input_subregion_name(geo_region_
 ↪name)
>>> print(valid_name)
Greater London

>>> geo_region_name = 'https://download.geofabrik.de/europe/great-britain.html'

>>> valid_name = geofabrik_downloader.validate_input_subregion_name(geo_region_
 ↪name)
>>> print(valid_name)
Great Britain
```

3.1.2 BBBikeDownloader

class pydriosm.downloader.**BBBikeDownloader**(*download_dir=None*)

Download OSM data from BBBike free download server.

Parameters **download_dir** – directory path to the downloaded file(s); if `None` (default), the current working directory

Variables

- ***Name*** (*str*) – name of data
- **Abbr** (*str*) – short name of the data
- ***URL*** (*str*) – URL of the homepage to the free download server
- **URLCities** (*str*) – URL of a list of cities available on the free download server
- **URLCitiesCoords** (*str*) – URL of coordinates of all the available cities
- **ValidFileFormats** (*list*) – valid file formats available on the free download server

Example:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> print(bbbike_downloader.Name)
BBBike OpenStreetMap data extracts

>>> print(bbbike_downloader.URL)
https://download.bbbike.org/osm/bbbike/
```

Methods

download_osm_data(subregion_names, ..., [, ...])	Download OSM data (in a specific file format) of one (or multiple) geographic region(s).
download_subregion_data(subregion_nar ...])	Download OSM data of all available formats for a geographic region.
file_exists(subregion_name, osm_file_format)	Check if a requested data file of a geographic region already exists locally, given its default filename.
get_coordinates_of_cities([update, ...])	Get location information of cities (geographic regions).
get_download_index([update, ...])	Get a dict-type index of available formats, data types and a download catalogue.
get_list_of_cities([update, ...])	Get a list of names of cities.
get_list_of_subregion_names([update, ...])	Get a list of names of all geographic regions.
get_subregion_catalogue([update, ...])	Get a catalogue for geographic regions.
get_subregion_download_catalogue(sub	Get a download catalogue of OSM data available for a geographic region.
get_subregion_download_url(subregion_ ...)	Get a valid URL for downloading OSM data of a specific file format for a geographic region.
get_valid_download_info(subregion_nar ...)	Get information of downloading (or downloaded) data file.
get_valid_file_formats()	Get a list of valid OSM data file formats.
validate_input_file_format(osm_file_f	Validate an input file format of OSM data.
validate_input_subregion_name(subreg	Validate an input name of a geographic region.

BBBikeDownloader.download_osm_data

BBBikeDownloader.**download_osm_data**(*subregion_names*, *osm_file_format*, *download_dir=None*, *update=False*, *confirmation_required=True*, *interval=None*, *verbose=False*, *ret_download_path=False*, ***kwargs*)

Download OSM data (in a specific file format) of one (or multiple) geographic region(s).

Parameters

- **subregion_names** (*str or list*) – name of a geographic region (or names of multiple geographic regions) available on BBBike free download server
- **osm_file_format** (*str*) – format (file extension) of an OSM data
- **download_dir** (*str or None*) – directory where downloaded OSM file is saved; if None (default), the default directory created by the package
- **update** (*bool*) – whether to update the data if it already exists, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **interval** (*int or None*) – interval (in second) between downloading two subregions, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **ret_download_path** (*bool*) – whether to return the path(s) to the downloaded file(s), defaults to False

Returns the path(s) to the downloaded file(s) when ret_download_path is True

Return type list or str

Examples:

```
>>> import os
>>> from pyhelpers.dir import delete_dir
>>> from pydriosm.downloader import BBBikeDownloader, cd_dat_bbbike

>>> bbbike_downloader = BBBikeDownloader()

>>> region_names = 'London'
>>> file_format = 'pbf'

>>> bbbike_downloader.download_osm_data(region_names, file_format, verbose=True)
To download .pbf data of the following geographic region(s):
    London
? [No]|Yes: yes
Downloading "London.osm.pbf" to "osm_bbbike\London\" ... Done.

>>> # Delete the created directory "osm_bbbike"
```

(continues on next page)

(continued from previous page)

```
>>> delete_dir(cd_dat_bbbike(), verbose=True)
The directory "osm_bbbike\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "osm_bbbike\" ... Done.

>>> region_names = ['leeds', 'birmingham']
>>> dwnld_dir = "tests"

>>> dwnld_paths = bbbike_downloader.download_osm_data(region_names, file_format,
...                                                   dwnld_dir, verbose=True,
...                                                   ret_download_path=True)
To download .pbf data of the following geographic region(s):
    Leeds
    Birmingham
? [No]|Yes: yes
Downloading "Leeds.osm.pbf" to "tests\" ... Done.
Downloading "Birmingham.osm.pbf" to "tests\" ... Done.

>>> for dwnld_path in dwnld_paths:
...     print(os.path.relpath(dwnld_path))
tests\Leeds.osm.pbf
tests\Birmingham.osm.pbf

>>> # Delete the above downloaded data files
>>> for dwnld_path in dwnld_paths:
...     os.remove(dwnld_path)
```

BBBikeDownloader.download_subregion_data

BBBikeDownloader.**download_subregion_data**(*subregion_name*, *download_dir=None*, *update=False*, *confirmation_required=True*, *interval=None*, *verbose=False*, *ret_download_path=False*, ***kwargs*)

Download OSM data of all available formats for a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region available on BBBike free download server
- **download_dir** (*str or None*) – directory where the downloaded file is saved, defaults to None
- **update** (*bool*) – whether to update the data if it already exists, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **interval** (*int or None*) – interval (in second) between downloading two subregions, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **ret_download_path** (*bool*) – whether to return the path(s) to the downloaded file(s), defaults to False

- **kwargs** – optional parameters of pyhelpers.ops.download_file_from_url()

Returns the path(s) to the downloaded file(s) when `ret_download_path` is `True`

Return type list or str

Example:

```
>>> import os
>>> from pyhelpers.dir import delete_dir
>>> from pydriosm.downloader import BBBikeDownloader, cd_dat_bbbike

>>> bbbike_downloader = BBBikeDownloader()

>>> # Download the BBBike OSM data of London
>>> region_name = 'london'

>>> bbbike_downloader.download_subregion_data(region_name, verbose=True)
To download all available BBBike OSM data of London
? [No]|Yes: yes
Downloading:
    London.osm.pbf ... Done.
    London.osm.gz ... Done.
    London.osm.shp.zip ... Done.
    London.osm.garmin-onroad-latin1.zip ... Done.
    London.osm.garmin-onroad.zip ... Done.
    London.osm.garmin-opentopo.zip ... Done.
    London.osm.garmin-osm.zip ... Done.
    London.osm.geojson.xz ... Done.
    London.osm.svg-osm.zip ... Done.
    London.osm.mapsforge-osm.zip ... Done.
    London.osm.navit.zip ... Done.
    London.osm.csv.xz ... Done.
    London.poly ... Done.
    CHECKSUM.txt ... Done.
Check out the downloaded OSM data at "osm_bbbike\London\".

>>> # Delete the download directory generated above
>>> delete_dir(cd_dat_bbbike(), verbose=True)
The directory "osm_bbbike\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "osm_bbbike\" ... Done.

>>> # Download the BBBike OSM data of Leeds
>>> region_name = 'leeds'
>>> dwnld_dir = "tests"

>>> dwnld_paths = bbbike_downloader.download_subregion_data(
...     region_name, dwnld_dir, confirmation_required=False, verbose=True,
...     ret_download_path=True)
Downloading all available BBBike OSM data of Leeds:
    Leeds.osm.pbf ... Done.
    Leeds.osm.gz ... Done.
    Leeds.osm.shp.zip ... Done.
    Leeds.osm.garmin-onroad-latin1.zip ... Done.
    Leeds.osm.garmin-onroad.zip ... Done.
    Leeds.osm.garmin-opentopo.zip ... Done.
    Leeds.osm.garmin-osm.zip ... Done.
    Leeds.osm.geojson.xz ... Done.
    Leeds.osm.svg-osm.zip ... Done.
```

(continues on next page)

(continued from previous page)

```
    Leeds.osm.mapsforge-osm.zip ... Done.
    Leeds.osm.navit.zip ... Done.
    Leeds.osm.csv.xz ... Done.
    Leeds.poly ... Done.
    CHECKSUM.txt ... Done.
Check out the downloaded OSM data at "tests\Leeds\".

>>> for dwnld_path in dwnld_paths:
...     print(os.path.relpath(dwnld_path))
tests\Leeds\Leeds.osm.pbf
tests\Leeds\Leeds.osm.gz
tests\Leeds\Leeds.osm.shp.zip
tests\Leeds\Leeds.osm.garmin-onroad-latin1.zip
tests\Leeds\Leeds.osm.garmin-onroad.zip
tests\Leeds\Leeds.osm.garmin-opentopo.zip
tests\Leeds\Leeds.osm.garmin-osm.zip
tests\Leeds\Leeds.osm.geojson.xz
tests\Leeds\Leeds.osm.svg-osm.zip
tests\Leeds\Leeds.osm.mapsforge-osm.zip
tests\Leeds\Leeds.osm.navit.zip
tests\Leeds\Leeds.osm.csv.xz
tests\Leeds\Leeds.poly
tests\Leeds\CHECKSUM.txt

>>> # Delete the download directory generated above
>>> delete_dir(os.path.commonpath(dwnld_paths), verbose=True)
The directory "tests\Leeds\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "tests\Leeds\" ... Done.
```

BBBikeDownloader.file_exists

BBBikeDownloader.**file_exists**(*subregion_name*, *osm_file_format*, *data_dir=None*, *update=False*, *verbose=False*, *ret_file_path=False*)

Check if a requested data file of a geographic region already exists locally, given its default filename.

Parameters

- **subregion_name** (*str*) – name of a geographic region available on BBBike free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **data_dir** (*str or None*) – directory for saving the downloaded file(s); if None (default), the default directory created by the package
- **update** (*bool*) – whether to (check on and) update the data, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False
- **ret_file_path** (*bool*) – whether to return the path to the data file (if it exists), defaults to False

Returns whether or not the requested data file exists; or the path to the data file

Return type bool or str

Examples:

```
>>> import os
>>> from pyhelpers.dir import delete_dir
>>> from pydriosm.downloader import BBBikeDownloader, cd_dat_bbbike

>>> bbbike_downloader = BBBikeDownloader()

>>> region_name = 'leeds'
>>> file_format = ".pbf"

>>> # Download the PBF data of London (to the default directory)
>>> bbbike_downloader.download_osm_data(region_name, file_format, verbose=True)
To download .pbf data of the following geographic region(s):
    Leeds
? [No]|Yes: yes
Downloading "Leeds.osm.pbf" to "osm_bbbike\Leeds\" ... Done.

>>> # Check whether the PBF data file exists; `ret_file_path` is by default `False`
>>> pbf_exists = bbbike_downloader.file_exists(region_name, file_format)

>>> type(pbf_exists)
bool
>>> # If the data file exists at the default directory created by the package
>>> print(pbf_exists)
True

>>> # Set `ret_file_path` to be `True`
>>> path_to_pbf = bbbike_downloader.file_exists(
...     subregion_name=region_name, osm_file_format=file_format, ret_file_
 ↪path=True)

>>> # If the data file exists at the default directory created by the package:
>>> type(path_to_pbf)
str
>>> print(os.path.relpath(path_to_pbf))
osm_bbbike\Leeds\Leeds.osm.pbf

>>> # Remove the directory or the PBF file and check again:
>>> delete_dir(cd_dat_bbbike(), confirmation_required=False, verbose=True)
Deleting "osm_bbbike\" ... Done.
>>> path_to_pbf = bbbike_downloader.file_exists(
...     subregion_name=region_name, osm_file_format=file_format, ret_file_
 ↪path=True)

>>> # Since the data file does not exist at the default directory
>>> type(path_to_pbf)
bool
>>> print(path_to_pbf)
False
```

BBBikeDownloader.get_coordinates_of_cities

BBBikeDownloader.**get_coordinates_of_cities**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get location information of cities (geographic regions).

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns location information of BBBike cities

Return type pandas.DataFrame or None

Example:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> # Location information of BBBike cities
>>> coords_of_cities = bbbike_downloader.get_coordinates_of_cities()

>>> type(coords_of_cities)
pandas.core.frame.DataFrame

>>> coords_of_cities.head()
          City  ... ur_latitude
0       Aachen  ...       50.99
1       Aarhus  ...      56.287
2     Adelaide  ...     -34.753
3  Albuquerque  ...     35.2173
4   Alexandria  ...       31.34
[5 rows x 13 columns]

>>> coords_of_cities.columns.tolist()
['City',
 'Real name',
 'Pref. language',
 'Local language',
 'Country',
 'Area/continent',
 'Population',
 'Step?',
 'Other cities',
 'll_longitude',
 'll_latitude',
 'ur_longitude',
 'ur_latitude']
```

BBBikeDownloader.get_download_index

BBBikeDownloader.**get_download_index**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get a dict-type index of available formats, data types and a download catalogue.

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns a list of available formats, a list of available data types and a dictionary of download catalogue

Return type dict

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> # Index for downloading OSM data available on the BBBike free download server
>>> dwnld_dict = bbbike_downloader.get_download_index()

>>> list(dwnld_dict.keys())
['FileFormat', 'DataType', 'Catalogue']

>>> catalogue = dwnld_dict['Catalogue']
>>> type(catalogue)
dict
>>> list(catalogue.keys())[:5]
['Aachen', 'Aarhus', 'Adelaide', 'Albuquerque', 'Alexandria']

>>> catalogue_leeds = catalogue['Leeds']
>>> type(catalogue_leeds)
pandas.core.frame.DataFrame
>>> catalogue_leeds.head()
                            Filename  ...          LastUpdate
0                       Leeds.osm.pbf  ... 2021-03-27 19:42:55
1                        Leeds.osm.gz  ... 2021-03-27 23:54:36
2                     Leeds.osm.shp.zip  ... 2021-03-28 00:08:26
3  Leeds.osm.garmin-onroad-latin1.zip  ... 2021-03-28 01:12:19
4          Leeds.osm.garmin-onroad.zip  ... 2021-03-28 01:11:50
[5 rows x 5 columns]
```

BBBikeDownloader.get_list_of_cities

BBBikeDownloader.**get_list_of_cities**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get a list of names of cities.

This is an alternative to *.get_list_of_subregion_names()*.

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns list of names of cities available on the BBBike free download server

Return type list or None

Example:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> # A list of BBBike cities' names
>>> names_of_cities = bbbike_downloader.get_list_of_cities()

>>> len(names_of_cities) > 200
True

>>> names_of_cities[:5]
['Heilbronn', 'Emden', 'Bremerhaven', 'Paris', 'Ostrava']

>>> names_of_cities[-5:]
['UlanBator', 'LaPaz', 'Sucre', 'Cusco', 'LaPlata']
```

BBBikeDownloader.get_list_of_subregion_names

BBBikeDownloader.**get_list_of_subregion_names**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get a list of names of all geographic regions.

This is an alternative to *.get_list_of_cities()*.

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to False
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Returns a list of geographic region names available on BBBike free download server

Return type list

Example:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> # A list of names of all BBBike geographic regions
>>> region_name_list = bbbike_downloader.get_list_of_subregion_names()

>>> len(region_name_list) > 200
True

>>> region_name_list[:5]
['Aachen', 'Aarhus', 'Adelaide', 'Albuquerque', 'Alexandria']

>>> region_name_list[-5:]
['Wroclaw', 'Wuerzburg', 'Wuppertal', 'Zagreb', 'Zuerich']
```

BBBikeDownloader.get_subregion_catalogue

BBBikeDownloader.**get_subregion_catalogue**(*update=False*, *confirmation_required=True*, *verbose=False*)

Get a catalogue for geographic regions.

Parameters

- **update** (*bool*) – whether to (check on and) update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns catalogue for subregions of BBBike data

Return type pandas.DataFrame or None

Example:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> # A BBBike catalogue of geographic regions
>>> subregion_catalog = bbbike_downloader.get_subregion_catalogue()

>>> type(subregion_catalog)
pandas.core.frame.DataFrame

>>> subregion_catalog.head()
          Name  ...                                                URL
1       Aachen  ...     https://download.bbbike.org/osm/bbbike/Aachen/
```

(continues on next page)

(continued from previous page)

```
2       Aarhus  ...     https://download.bbbike.org/osm/bbbike/Aarhus/
3     Adelaide  ...   https://download.bbbike.org/osm/bbbike/Adelaide/
4  Albuquerque  ...  https://download.bbbike.org/osm/bbbike/Albuque...
5   Alexandria  ...  https://download.bbbike.org/osm/bbbike/Alexand...
[5 rows x 3 columns]

>>> subregion_catalog.columns.to_list()
['Name', 'Last Modified', 'URL']
```

BBBikeDownloader.get_subregion_download_catalogue

BBBikeDownloader.**get_subregion_download_catalogue**(*subregion_name*, *confirmation_required=True*, *verbose=False*)

Get a download catalogue of OSM data available for a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region available on BBBike free download server
- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

Returns a catalogues for subregion downloads

Return type pandas.DataFrame or None

Example:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> region_name = 'leeds'

>>> # A download catalogue for Leeds
>>> leeds_dwnld_cat = bbbike_downloader.get_subregion_download_catalogue(
...     subregion_name=region_name, verbose=True)
To collect the download catalogue for "Leeds"
? [No]|Yes: yes
Collecting the data ... Done.

>>> leeds_dwnld_cat.head()
                           Filename  ...          LastUpdate
0                      Leeds.osm.pbf  ... 2020-09-25 10:04:25
1                       Leeds.osm.gz  ... 2020-09-25 15:11:49
2                  Leeds.osm.shp.zip  ... 2020-09-25 15:33:10
3  Leeds.osm.garmin-onroad-latin1.zip  ... 2020-09-25 17:49:15
4        Leeds.osm.garmin-onroad.zip  ... 2020-09-25 17:49:04
[5 rows x 5 columns]

>>> leeds_dwnld_cat.columns.tolist()
['Filename', 'URL', 'DataType', 'Size', 'LastUpdate']
```

BBBikeDownloader.get_subregion_download_url

BBBikeDownloader.**get_subregion_download_url**(*subregion_name*, *osm_file_format*)

Get a valid URL for downloading OSM data of a specific file format for a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region available on BBBike free download server
- **osm_file_format** (*str*) – format (file extension) of an OSM data

Returns a valid name of `subregion_name` and a download URL for the given `osm_file_format`

Return type tuple

Examples:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> region_name = 'leeds'
>>> file_format = 'pbf'

>>> # Get a valid subregion name and its download URL
>>> rn, dl = bbbike_downloader.get_subregion_download_url(region_name, file_format)

>>> print(rn)
Leeds
>>> print(dl)
https://download.bbbike.org/osm/bbbike/Leeds/Leeds.osm.pbf

>>> file_format = 'csv.xz'
>>> rn, dl = bbbike_downloader.get_subregion_download_url(region_name, file_format)

>>> print(rn)
Leeds
>>> print(dl)
https://download.bbbike.org/osm/bbbike/Leeds/Leeds.osm.csv.xz
```

BBBikeDownloader.get_valid_download_info

BBBikeDownloader.**get_valid_download_info**(*subregion_name*, *osm_file_format*, *download_dir=None*, *mkdir=False*)

Get information of downloading (or downloaded) data file.

The information includes a valid subregion name, a default filename, a URL and an absolute path where the data file is (to be) saved locally.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive)
- **osm_file_format** (*str*) – format (file extension) of an OSM data

- **download_dir** (*str or None*) – directory where downloaded OSM file is saved; if None (default), the default directory created by the package
- **mkdir** (*bool*) – whether to create a directory, defaults to False

Returns valid subregion name, filename, download url and absolute file path

Return type tuple

Examples:

```
>>> import os
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> region_name = 'leeds'
>>> file_format = 'pbf'

>>> # valid subregion name, filename, download url and absolute file path
>>> info = bbbike_downloader.get_valid_download_info(region_name, file_format)
>>> sub_reg_name, pbf_filename, dwnld_url, path_to_pbf = info

>>> print(sub_reg_name)
Leeds
>>> print(pbf_filename)
Leeds.osm.pbf
>>> print(dwnld_url)
https://download.bbbike.org/osm/bbbike/Leeds/Leeds.osm.pbf
>>> print(os.path.relpath(path_to_pbf))
osm_bbbike\Leeds\Leeds.osm.pbf
```

BBBikeDownloader.get_valid_file_formats

BBBikeDownloader.**get_valid_file_formats**()

Get a list of valid OSM data file formats.

Returns a list of valid BBBike OSM file formats on BBBike free download server

Return type list

Example:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> file_formats = bbbike_downloader.get_valid_file_formats()

>>> for file_format in file_formats:
...     print(file_format)
.pbf
.gz
.shp.zip
.garmin-onroad-latin1.zip
.garmin-onroad.zip
.garmin-opentopo.zip
```

(continues on next page)

(continued from previous page)

```
.garmin-osm.zip
.geojson.xz
.svg-osm.zip
.mapsforge-osm.zip
.navit.zip
.csv.xz
```

BBBikeDownloader.validate_input_file_format

BBBikeDownloader.**validate_input_file_format**(*osm_file_format*)

Validate an input file format of OSM data.

The validation is done by matching the input `osm_file_format` to a filename extension available on BBBike free download server.

Parameters **osm_file_format** (*str*) – file extension of an OSM data extract

Returns valid file format (file extension)

Return type str

Example:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()

>>> file_format = 'PBF'

>>> valid_file_format = bbbike_downloader.validate_input_file_format(file_format)

>>> print(valid_file_format)
.pbf
```

BBBikeDownloader.validate_input_subregion_name

BBBikeDownloader.**validate_input_subregion_name**(*subregion_name*)

Validate an input name of a geographic region.

The validation is done by matching the input `subregion_name` to a name of a geographic region available on BBBike free download server.

Parameters **subregion_name** (*str*) – name of a geographic region (case-insensitive)

Returns valid subregion name that matches, or is the most similar to, the input `subregion_name`

Return type str

Example:

```
>>> from pydriosm.downloader import BBBikeDownloader

>>> bbbike_downloader = BBBikeDownloader()
```

(continues on next page)

(continued from previous page)

```
>>> region_name = 'leeds'

>>> valid_name = bbbike_downloader.validate_input_subregion_name(region_name)

>>> print(valid_name)
Leeds
```

3.2 reader

Read OpenStreetMap (OSM) data extracts.

Classes

GeofabrikReader([max_tmpfile_size, data_dir])	Read Geofabrik data extracts.
BBBikeReader([max_tmpfile_size, data_dir])	Read BBBike data extracts.

3.2.1 GeofabrikReader

class pydriosm.reader.**GeofabrikReader**(*max_tmpfile_size=5000*, *data_dir=None*)

Read Geofabrik data extracts.

Parameters

- **max_tmpfile_size** (*int or None*) – defaults to 5000, see also *gdal_configurations()*
- **data_dir** (*str or None*) – (a path or a name of) a directory where a data file is; if None (default), a folder osm_geofabrik under the current working directory

Variables

- ***Downloader*** (GeofabrikDownloader) – instance of the class *GeofabrikDownloader*
- ***Name*** (*str*) – name of the data resource
- ***URL*** (*str*) – URL of the homepage to the Geofabrik free download server

Example:

```
>>> from pydriosm.reader import GeofabrikReader

>>> geofabrik_reader = GeofabrikReader()

>>> print(geofabrik_reader.Name)
Geofabrik OpenStreetMap data extracts
```

Methods

get_osm_pbf_layer_names(subregion_nar ...])	Get indices and names of all layers in the PBF data file of a given (sub)region.
get_path_to_osm_file(subregion_name, ...[, ...])	Get the local path to an OSM data file of a geographic region.
get_path_to_osm_shp(subregion_name[, ...])	Get path(s) to .shp file(s) for a geographic region (by searching a local data directory).
merge_subregion_layer_shp(subregion_r ...)	Merge shapefiles for a specific layer of two or multiple geographic regions.
read_osm_pbf(subregion_name[, data_dir, ...])	Read a PBF (.osm.pbf) data file of a geographic region.
read_shp_zip(subregion_name[, layer_names, ...])	Read a .shp.zip data file of a geographic region.

GeofabrikReader.get_osm_pbf_layer_names

GeofabrikReader.**get_osm_pbf_layer_names**(*subregion_name*, *data_dir=None*)

Get indices and names of all layers in the PBF data file of a given (sub)region.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on Geofabrik free download server
- **data_dir** –

Returns indices and names of each layer of the PBF data file

Return type dict

Example:

```
>>> import os
>>> from pydriosm.reader import GeofabrikReader

>>> geofabrik_reader = GeofabrikReader()

>>> # Download the PBF data file of Rutland to "tests\"
>>> path_to_rutland_pbf = geofabrik_reader.Downloader.download_osm_data(
...     subregion_names='Rutland', osm_file_format=".pbf", download_dir="tests",
...     confirmation_required=False, ret_download_path=True)

>>> lyr_idx_names = geofabrik_reader.get_osm_pbf_layer_names(path_to_rutland_pbf)

>>> lyr_idx_names
{0: 'points',
 1: 'lines',
 2: 'multilinestrings',
 3: 'multipolygons',
 4: 'other_relations'}
```

GeofabrikReader.get_path_to_osm_file

GeofabrikReader.**get_path_to_osm_file**(*subregion_name*, *osm_file_format*, *data_dir=None*)

Get the local path to an OSM data file of a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on Geofabrik free download server
- **osm_file_format** (*str*) – file format of the OSM data available on the free download server
- **data_dir** (*str or None*) – directory where the data file of the subregion_name is located/saved; if None (default), the default local directory

Returns path to PBF (.osm.pbf) file

Return type str or None

Example:

```
>>> import os
>>> from pydriosm.reader import GeofabrikReader

>>> geofabrik_reader = GeofabrikReader()

>>> region_name = 'Rutland'
>>> file_format = ".pbf"

>>> path_to_rutland_pbf = geofabrik_reader.get_path_to_osm_file(
...     region_name, file_format)

>>> print(path_to_rutland_pbf)
# (if "rutland-latest.osm.pbf" is unavailable at the package data directory)
# None

>>> # Specify a download directory
>>> dwnld_dir = "tests"

>>> # Download the PBF data file of Rutland to "tests\"
>>> geofabrik_reader.Downloader.download_osm_data(
...     region_name, file_format, download_dir=dwnld_dir, verbose=True)
To download .osm.pbf data of the following geographic region(s):
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "tests\" ... Done.

>>> path_to_rutland_pbf = geofabrik_reader.get_path_to_osm_file(
...     region_name, file_format, data_dir=dwnld_dir)
>>> print(os.path.relpath(path_to_rutland_pbf))
tests\rutland-latest.osm.pbf

>>> # Delete the downloaded PBF data file
>>> os.remove(path_to_rutland_pbf)
```

GeofabrikReader.get_path_to_osm_shp

GeofabrikReader.**get_path_to_osm_shp**(*subregion_name*, *layer_name=None*, *feature_name=None*, *data_dir=None*, *file_ext='.shp'*)

Get path(s) to .shp file(s) for a geographic region (by searching a local data directory).

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on Geofabrik free download server
- **layer_name** (*str or None*) – name of a .shp layer (e.g. `'railways'`), defaults to `None`
- **feature_name** (*str or None*) – name of a feature (e.g. `'rail'`); if `None` (default), all available features included
- **data_dir** (*str or None*) – directory where the search is conducted; if `None` (default), the default directory
- **file_ext** (*str*) – file extension, defaults to `".shp"`

Returns path(s) to .shp file(s)

Return type list or str

Examples:

```
>>> import os
>>> from pyhelpers.dir import delete_dir
>>> from pydriosm.reader import GeofabrikReader, unzip_shp_zip, parse_layer_shp

>>> geofabrik_reader = GeofabrikReader()

>>> region_name = 'Rutland'
>>> file_format = ".shp"

>>> # (if "gis.osm_railways_free_1.shp" is unavailable)
>>> path_to_shp_file = geofabrik_reader.get_path_to_osm_shp(region_name)
>>> print(path_to_shp_file)
[]

>>> dwnld_dir = "tests"

>>> # Download the shapefiles of Rutland
>>> path_to_rutland_shp_zip = geofabrik_reader.Downloader.download_osm_data(
...     region_name, file_format, dwnld_dir, confirmation_required=False,
...     ret_download_path=True)

>>> # Extract the downloaded .zip file
>>> unzip_shp_zip(path_to_rutland_shp_zip, verbose=True)
Extracting "tests\rutland-latest-free.shp.zip" ...
to "tests\rutland-latest-free-shp\"
Done.

>>> lyr_name = 'railways'

>>> # Get the file path of 'railways' shapefile
>>> path_to_rutland_railways_shp = geofabrik_reader.get_path_to_osm_shp(
```

(continues on next page)

(continued from previous page)

```
...     region_name, lyr_name, data_dir=dwnld_dir)

>>> print(os.path.relpath(path_to_rutland_railways_shp))
tests\rutland-latest-free-shp\gis_osm_railways_free_1.shp

>>> feat_name = 'rail'

>>> # Get/save shapefile data of features labelled 'rail' only
>>> _ = parse_layer_shp(path_to_rutland_railways_shp, feature_names=feat_name,
...                     save_fclass_shp=True)

>>> # Get the file path to the data of 'rail'
>>> path_to_rutland_railways_rail_shp = geofabrik_reader.get_path_to_osm_shp(
...     region_name, lyr_name, feat_name, data_dir=dwnld_dir)

>>> print(os.path.relpath(path_to_rutland_railways_rail_shp))
tests\rutland-latest-free-shp\railways\gis_osm_railways_free_1_rail.shp

>>> # Retrieve the data of 'rail' feature
>>> rutland_railways_rail_shp = parse_layer_shp(path_to_rutland_railways_rail_shp)

>>> rutland_railways_rail_shp.head()
    osm_id  code  ...                                        coordinates shape_type
0  2162114  6101  ...  [(-0.4528083, 52.6993402), (-0.4518933, 52.698...          3
1  3681043  6101  ...  [(-0.6531215, 52.5730787), (-0.6531793, 52.572...          3
2  3693985  6101  ...  [(-0.7323403, 52.6782102), (-0.7319059, 52.678...          3
3  3693986  6101  ...  [(-0.6173072, 52.6132317), (-0.6241869, 52.614...          3
4  4806329  6101  ...  [(-0.4576926, 52.7035194), (-0.4565358, 52.702...          3
[5 rows x 9 columns]

>>> # Delete the extracted files
>>> delete_dir(os.path.dirname(path_to_rutland_railways_shp), verbose=True)
The directory "tests\rutland-latest-free-shp\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "tests\rutland-latest-free-shp\" ... Done.

>>> # Delete the downloaded .shp.zip file
>>> os.remove(path_to_rutland_shp_zip)
```

GeofabrikReader.merge_subregion_layer_shp

GeofabrikReader.**merge_subregion_layer_shp**(*subregion_names*, *layer_name*, *data_dir=None*, *method='pyshp'*, *update=False*, *download_confirmation_required=True*, *rm_zip_extracts=True*, *merged_shp_dir=None*, *rm_shp_temp=True*, *verbose=False*, *ret_merged_shp_path=False*)

Merge shapefiles for a specific layer of two or multiple geographic regions.

Parameters

- **subregion_names** (*list*) – names of geographic region (case-insensitive) that is available on Geofabrik free download server
- **layer_name** (*str*) – name of a layer (e.g. 'railways')

- **method** (*str*) – the method used to merge/save shapefiles; options include: `'pyshp'` (default) and `'geopandas'` (or `'gpd'`) if `method='geopandas'`, this function relies on geopandas.GeoDataFrame.to_file(); otherwise, it by default uses shapefile.Writer()
- **update** (*bool*) – whether to update the source .shp.zip files, defaults to `False`
- **download_confirmation_required** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to `True`
- **data_dir** (*str or None*) – directory where the .shp.zip data files are located/saved; if `None` (default), the default directory
- **rm_zip_extracts** (*bool*) – whether to delete the extracted files, defaults to `False`
- **rm_shp_temp** (*bool*) – whether to delete temporary layer files, defaults to `False`
- **merged_shp_dir** (*str or None*) – if `None` (default), use the layer name as the name of the folder where the merged .shp files will be saved
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **ret_merged_shp_path** (*bool*) – whether to return the path to the merged .shp file, defaults to `False`

Returns the path to the merged file when `ret_merged_shp_path=True`

Return type list or str

Examples:

```
>>> import os
>>> from pyhelpers.dir import cd, delete_dir
>>> from pydriosm.reader import GeofabrikReader, read_shp_file

>>> geofabrik_reader = GeofabrikReader()

>>> # -- Example 1 ------------------------------------------------------------

>>> # To merge 'railways' of Greater Manchester and West Yorkshire
>>> sr_names = ['Manchester', 'West Yorkshire']
>>> lyr_name = 'railways'
>>> dat_dir = "tests"

>>> path_to_merged_shp_file = geofabrik_reader.merge_subregion_layer_shp(
...     sr_names, lyr_name, dat_dir, verbose=True, ret_merged_shp_path=True)
Downloading "greater-manchester-latest-free.shp.zip" to "tests\" ... Done.
Downloading "west-yorkshire-latest-free.shp.zip" to "tests\" ... Done.
Extracting the following layer(s):
    'railways'
from "tests\greater-manchester-latest-free.shp.zip" ...
to "tests\greater-manchester-latest-free-shp\"
Done.
Extracting the following layer(s):
    'railways'
```

(continues on next page)

(continued from previous page)

```
from "tests\west-yorkshire-latest-free.shp.zip" ...
to "tests\west-yorkshire-latest-free-shp\"
Done.
Merging the following shapefiles:
    "greater-manchester_gis_osm_railways_free_1.shp"
    "west-yorkshire_gis_osm_railways_free_1.shp"
In progress ... Done.
Find the merged shapefile at "tests\greater-manchester_west-yorkshire_railways\".

>>> print(os.path.relpath(path_to_merged_shp_file))
tests\...\greater-manchester_west-yorkshire_railways.shp

>>> # Read the merged data
>>> manchester_yorkshire_railways_shp = read_shp_file(path_to_merged_shp_file)

>>> manchester_yorkshire_railways_shp.head()
    osm_id  code  ...                                        coordinates shape_type
0   928999  6101  ...  [(-2.2844594, 53.4802681), (-2.2851997, 53.480...          3
1   929904  6101  ...  [(-2.2919566, 53.4619298), (-2.2924877, 53.461...          3
2   929905  6102  ...  [(-2.2794048, 53.4605819), (-2.2799773, 53.460...          3
3  3663332  6102  ...  [(-2.2382517, 53.4818141), (-2.2381708, 53.481...          3
4  3996086  6101  ...  [(-2.6003908, 53.4602313), (-2.6009371, 53.459...          3
[5 rows x 9 columns]

>>> # Delete the merged files
>>> delete_dir(os.path.dirname(path_to_merged_shp_file), verbose=True)
The directory "tests\greater-manchester_west-yorkshire_railways" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "tests\greater-manchester_west-yorkshire_railways" ... Done.

>>> # Delete the downloaded .shp.zip data files
>>> os.remove(cd(dat_dir, "greater-manchester-latest-free.shp.zip"))
>>> os.remove(cd(dat_dir, "west-yorkshire-latest-free.shp.zip"))

>>> # -- Example 2 ------------------------------------------------------------

>>> # To merge 'transport' of Greater London, Kent and Surrey

>>> sr_names = ['London', 'Kent', 'Surrey']
>>> lyr_name = 'transport'

>>> path_to_merged_shp_file = geofabrik_reader.merge_subregion_layer_shp(
...     sr_names, lyr_name, dat_dir, verbose=True, ret_merged_shp_path=True)
To download .shp.zip data of the following geographic region(s):
    Greater London
    Kent
    Surrey
? [No]|Yes: yes
Downloading "greater-london-latest-free.shp.zip" to "tests\" ... Done.
Downloading "kent-latest-free.shp.zip" to "tests\" ... Done.
Downloading "surrey-latest-free.shp.zip" to "tests\" ... Done.
Extracting the following layer(s):
    'transport'
from "tests\greater-london-latest-free.shp.zip" ...
to "tests\greater-london-latest-free-shp\"
Done.
Extracting the following layer(s):
    'transport'
from "tests\kent-latest-free.shp.zip" ...
to "tests\kent-latest-free-shp\"
```

(continues on next page)

(continued from previous page)

```
Done.
Extracting the following layer(s):
    'transport'
from "tests\surrey-latest-free.shp.zip" ...
to "tests\surrey-latest-free-shp\"
Done.
Merging the following shapefiles:
    "greater-london_gis_osm_transport_a_free_1.shp"
    "greater-london_gis_osm_transport_free_1.shp"
    "kent_gis_osm_transport_a_free_1.shp"
    "kent_gis_osm_transport_free_1.shp"
    "surrey_gis_osm_transport_a_free_1.shp"
    "surrey_gis_osm_transport_free_1.shp"
In progress ... Done.
Find the merged .shp file(s) at "tests\greater-london_kent_surrey_transport\".

>>> print(os.path.relpath(path_to_merged_shp_file))
tests\...\greater-london_kent_surrey_transport.shp

>>> # Read the merged shapefile
>>> merged_transport_shp = read_shp_file(path_to_merged_shp_file)

>>> merged_transport_shp.head()
     osm_id  ...  shape_type
0   5077928  ...           5
1   8610280  ...           5
2  15705264  ...           5
3  23077379  ...           5
4  24016945  ...           5
[5 rows x 6 columns]

>>> # Delete the merged files
>>> delete_dir(os.path.dirname(path_to_merged_shp_file), verbose=True)
The directory "tests\greater-london_kent_surrey_transport\" is not empty.
Confirmed to delete it? [No]|Yes: >? yes
Deleting "tests\greater-london_kent_surrey_transport\" ... Done.

>>> # Delete the downloaded .shp.zip data files
>>> os.remove(cd(dat_dir, "greater-london-latest-free.shp.zip"))
>>> os.remove(cd(dat_dir, "kent-latest-free.shp.zip"))
>>> os.remove(cd(dat_dir, "surrey-latest-free.shp.zip"))
```

GeofabrikReader.read_osm_pbf

GeofabrikReader.**read_osm_pbf**(*subregion_name*, *data_dir=None*, *chunk_size_limit=50*, *parse_raw_feat=False*, *transform_geom=False*, *transform_other_tags=False*, *update=False*, *download_confirmation_required=True*, *pickle_it=False*, *ret_pickle_path=False*, *rm_osm_pbf=False*, *verbose=False*, ***kwargs*)

Read a PBF (.osm.pbf) data file of a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on Geofabrik free download server
- **data_dir** (*str or None*) – directory where the .osm.pbf data file is

located/saved; if None, the default local directory

- **chunk_size_limit** (*int*) – threshold (in MB) that triggers the use of chunk parser, defaults to 50; if the size of the .osm.pbf file (in MB) is greater than chunk_size_limit, it will be parsed in a chunk-wise way
- **parse_raw_feat** (*bool*) – whether to parse each feature in the raw data, defaults to False
- **transform_geom** (*bool*) – whether to transform a single coordinate (or a collection of coordinates) into a geometric object, defaults to False
- **transform_other_tags** (*bool*) – whether to transform a 'other_tags' into a dictionary, defaults to False
- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to False
- **download_confirmation_required** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to True
- **pickle_it** (*bool*) – whether to save the .pbf data as a .pickle file, defaults to False
- **ret_pickle_path** (*bool*) – (when pickle_it=True) whether to return a path to the saved pickle file
- **rm_osm_pbf** (*bool*) – whether to delete the downloaded .osm.pbf file, defaults to False
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False
- **kwargs** – optional parameters of *parse_osm_pbf()*

Returns dictionary of the .osm.pbf data; when pickle_it=True, return a tuple of the dictionary and a path to the pickle file

Return type dict or tuple or None

Examples:

```
>>> import os
>>> from pydriosm.reader import GeofabrikReader

>>> geofabrik_reader = GeofabrikReader()

>>> sr_name = 'Rutland'
>>> dat_dir = "tests"

>>> # If the PBF data of Rutland is not available at the specified data directory,
>>> # the function may ask whether to download the latest data
>>> rutland_pbf_raw = geofabrik_reader.read_osm_pbf(sr_name, dat_dir, verbose=True)
To download .osm.pbf data of the following geographic region(s):
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "tests\" ... Done.

>>> list(rutland_pbf_raw.keys())
```

(continues on next page)

(continued from previous page)

```
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> rutland_pbf_raw_points = rutland_pbf_raw['points']
>>> rutland_pbf_raw_points.head()
                                              points
0  {"type": "Feature", "geometry": {"type": "Poin...
1  {"type": "Feature", "geometry": {"type": "Poin...
2  {"type": "Feature", "geometry": {"type": "Poin...
3  {"type": "Feature", "geometry": {"type": "Poin...
4  {"type": "Feature", "geometry": {"type": "Poin...

>>> # Set `parse_raw_feat` to be True
>>> rutland_pbf_parsed = geofabrik_reader.read_osm_pbf(sr_name, dat_dir,
...                                                    parse_raw_feat=True,
...                                                    verbose=True)
Parsing "tests\rutland-latest.osm.pbf" ... Done.

>>> rutland_pbf_parsed_points = rutland_pbf_parsed['points']
>>> rutland_pbf_parsed_points.head()
         id               coordinates  ...                    other_tags
0    488432  [-0.5134241, 52.6555853]  ...                "odbl"=>"clean"
1    488658  [-0.5313354, 52.6737716]  ...                           None
2  13883868  [-0.7229332, 52.5889864]  ...                           None
3  14049101  [-0.7249922, 52.6748223]  ...  "traffic_calming"=>"cushion"
4  14558402  [-0.7266686, 52.6695051]  ...      "direction"=>"clockwise"
[5 rows x 12 columns]

>>> # Set both `parse_raw_feat` and `transform_geom` to be True
>>> rutland_pbf_parsed_1 = geofabrik_reader.read_osm_pbf(sr_name, dat_dir,
...                                                      parse_raw_feat=True,
...                                                      transform_geom=True,
...                                                      verbose=True)
Parsing "tests\rutland-latest.osm.pbf" ... Done.

>>> rutland_pbf_parsed_1_points = rutland_pbf_parsed_1['points']
>>> rutland_pbf_parsed_1_points[['coordinates', 'other_tags']].head()
                            coordinates                    other_tags
0           POINT (-0.5134241 52.6555853)                "odbl"=>"clean"
1           POINT (-0.5313354 52.6737716)                           None
2  POINT (-0.7229332000000001 52.5889864)                           None
3           POINT (-0.7249816 52.6748426)  "traffic_calming"=>"cushion"
4           POINT (-0.7266581 52.6695058)      "direction"=>"clockwise"

>>> # Set `parse_raw_feat` `transform_geom` and `transform_other_tags` to be True
>>> rutland_pbf_parsed_2 = geofabrik_reader.read_osm_pbf(sr_name, dat_dir,
...                                                      parse_raw_feat=True,
...                                                      transform_geom=True,
...                                                      transform_other_tags=True,
...                                                      verbose=True)
Parsing "tests\rutland-latest.osm.pbf" ... Done.

>>> rutland_pbf_parsed_2_points = rutland_pbf_parsed_2['points']
>>> rutland_pbf_parsed_2_points[['coordinates', 'other_tags']].head()
                            coordinates                      other_tags
0           POINT (-0.5134241 52.6555853)                {'odbl': 'clean'}
1           POINT (-0.5313354 52.6737716)                             None
2  POINT (-0.7229332000000001 52.5889864)                             None
3           POINT (-0.7249816 52.6748426)  {'traffic_calming': 'cushion'}
4           POINT (-0.7266581 52.6695058)      {'direction': 'clockwise'}
```

(continues on next page)

(continued from previous page)

```
>>> # Delete the downloaded PBF data file
>>> os.remove(os.path.join(dat_dir, "rutland-latest.osm.pbf"))
```

GeofabrikReader.read_shp_zip

GeofabrikReader.**read_shp_zip**(*subregion_name*, *layer_names=None*, *feature_names=None*, *data_dir=None*, *update=False*, *download_confirmation_required=True*, *pickle_it=False*, *ret_pickle_path=False*, *rm_extracts=False*, *rm_shp_zip=False*, *verbose=False*)

Read a .shp.zip data file of a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on Geofabrik free download server
- **layer_names** (*str or list or None*) – name of a .shp layer, e.g. 'railways', or names of multiple layers; if `None` (default), all available layers
- **feature_names** (*str or list or None*) – name of a feature, e.g. 'rail', or names of multiple features; if `None` (default), all available features
- **data_dir** (*str or None*) – directory where the .shp.zip data file is located/saved; if `None`, the default directory
- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to `False`
- **download_confirmation_required** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to `True`
- **pickle_it** (*bool*) – whether to save the .shp data as a .pickle file, defaults to `False`
- **ret_pickle_path** (*bool*) – (when `pickle_it=True`) whether to return a path to the saved pickle file
- **rm_extracts** (*bool*) – whether to delete extracted files from the .shp.zip file, defaults to `False`
- **rm_shp_zip** (*bool*) – whether to delete the downloaded .shp.zip file, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns dictionary of the shapefile data, with keys and values being layer names and tabular data (in the format of geopandas.GeoDataFrame), respectively

Return type dict or None

Example:

```
>>> from pydriosm.reader import GeofabrikReader

>>> geofabrik_reader = GeofabrikReader()

>>> sr_name = 'Rutland'
>>> dat_dir = "tests"

>>> rutland_shp = geofabrik_reader.read_shp_zip(
...     subregion_name=sr_name, data_dir=dat_dir, verbose=True)
To download .shp.zip data of the following geographic region(s):
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest-free.shp.zip" to "tests\" ... Done.
Extracting "tests\rutland-latest-free.shp.zip" ...
to "tests\rutland-latest-free-shp\"
Done.

>>> list(rutland_shp.keys())
['buildings',
 'traffic',
 'water',
 'roads',
 'places',
 'pofw',
 'waterways',
 'pois',
 'landuse',
 'transport',
 'natural',
 'railways']

>>> # Data of the 'railways' layer
>>> rutland_shp_railways = rutland_shp['railways']
>>> rutland_shp_railways.head()
    osm_id  code  ...                                        coordinates shape_type
0  2162114  6101  ...  [(-0.4528083, 52.6993402), (-0.4518933, 52.698...          3
1  3681043  6101  ...  [(-0.6531215, 52.5730787), (-0.6531793, 52.572...          3
2  3693985  6101  ...  [(-0.7323403, 52.6782102), (-0.7319059, 52.678...          3
3  3693986  6101  ...  [(-0.6173072, 52.6132317), (-0.6241869, 52.614...          3
4  4806329  6101  ...  [(-0.4576926, 52.7035194), (-0.4565358, 52.702...          3
[5 rows x 9 columns]

>>> # Read data of the 'transport' layer only from the original .shp.zip file
>>> # (and delete any extracts)
>>> sr_layer = 'transport'

>>> rutland_shp_transport = geofabrik_reader.read_shp_zip(sr_name, sr_layer,
...                                                       data_dir=dat_dir,
...                                                       verbose=True,
...                                                       rm_extracts=True)
Deleting the extracts "tests\rutland-latest-free-shp\"  ... Done.

>>> list(rutland_shp_transport.keys())
['transport']

>>> rutland_shp_transport['transport'].head()
      osm_id  ...  shape_type
0  232038062  ...           5
1  468873547  ...           5
2  468873548  ...           5
3  468873553  ...           5
```

(continues on next page)

(continued from previous page)

```
4  468873559  ...           5
[5 rows x 6 columns]

>>> # Read data of only the 'bus_stop' feature (in the 'transport' layer)
>>> # from the original .shp.zip file (and delete any extracts)
>>> feat_name = 'bus_stop'

>>> rutland_bus_stop = geofabrik_reader.read_shp_zip(sr_name, sr_layer, feat_name,
...                                                  dat_dir, verbose=True,
...                                                  rm_extracts=True)
Extracting the following layer(s):
    'transport'
from "tests\rutland-latest-free.shp.zip" ...
to "tests\rutland-latest-free-shp\"
Done.
Deleting the extracts "tests\rutland-latest-free-shp\"  ... Done.

>>> list(rutland_bus_stop.keys())
['transport']

>>> print(rutland_bus_stop['transport'].fclass.unique())
['bus_stop']

>>> # Read multiple features of multiple layers
>>> # (and delete both the original .shp.zip file and extracts)
>>> sr_layers = ['traffic', 'roads']
>>> feat_names = ['parking', 'trunk']

>>> rutland_shp_tr_pt = geofabrik_reader.read_shp_zip(sr_name, sr_layers, feat_
 ↪name,
...                                                   dat_dir, verbose=True,
...                                                   rm_extracts=True,
...                                                   rm_shp_zip=True)
Extracting the following layer(s):
    'traffic'
    'roads'
from "tests\rutland-latest-free.shp.zip" ...
to "tests\rutland-latest-free-shp\"
Done.
Deleting the extracts "tests\rutland-latest-free-shp\"  ... Done.
Deleting "tests\rutland-latest-free.shp.zip" ... Done.

>>> list(rutland_shp_tr_pt.keys())
['traffic', 'roads']

>>> # Data of the 'traffic' layer
>>> rutland_shp_tr_pt_traffic = rutland_shp_tr_pt['traffic']
>>> rutland_shp_tr_pt_traffic.head()
         osm_id  code  ...                  coordinates shape_type
204    14558402  5202  ...  [[-0.7266581, 52.6695058]]          1
206    14583750  5202  ...  [[-0.4704691, 52.6548803]]          1
213    18335108  5202  ...  [[-0.7384552, 52.6674072]]          1
255   862120532  5202  ...  [[-0.7320612, 52.6688328]]          1
291  1584865939  5202  ...   [[-0.7391079, 52.674775]]          1
[5 rows x 6 columns]

>>> # Data of the 'roads' layer
>>> rutland_shp_tr_pt_roads = rutland_shp_tr_pt['roads']
>>> rutland_shp_tr_pt_roads.head()
         osm_id  ...  shape_type
```

(continues on next page)

(continued from previous page)

```
1320   73599134  ...           3
1321   73599136  ...           3
1557  101044857  ...           3
1561  101044867  ...           3
1682  101326487  ...           3
[5 rows x 12 columns]
```

Attributes

DataDir

GeofabrikReader.DataDir

property GeofabrikReader.**DataDir**

3.2.2 BBBikeReader

class pydriosm.reader.**BBBikeReader**(*max_tmpfile_size=5000*, *data_dir=None*)

Read BBBike data extracts.

Parameters

- **max_tmpfile_size** (*int or None*) – defaults to 5000, see also *gdal_configurations()*
- **data_dir** (*str or None*) – (a path or a name of) a directory where a data file is; if None (default), a folder osm_bbbike under the current working directory

Variables

- ***Downloader*** (BBBikeDownloader) – instance of the class *BBBikeDownloader*
- ***Name*** (*str*) – name of the data resource
- ***URL*** (*str*) – URL of the homepage to the BBBike free download server

Example:

```
>>> from pydriosm.reader import BBBikeReader

>>> bbbike_reader = BBBikeReader()

>>> print(bbbike_reader.Name)
BBBike OpenStreetMap data extracts
```

Methods

get_path_to_osm_file(subregion_name, ...[, ...])	Get the path to an OSM data file (if available) of a specific file format for a geographic region.
read_csv_xz(subregion_name[, data_dir, ...])	Read a compressed CSV (.csv.xz) data file of a geographic region.
read_geojson_xz(subregion_name[, data_dir, ...])	Read a .geojson.xz data file of a geographic region.
read_osm_pbf(subregion_name[, data_dir, ...])	Read a PBF data file of a geographic region.
read_shp_zip(subregion_name[, layer_names, ...])	Read a shapefile of a geographic region.

BBBikeReader.get_path_to_osm_file

BBBikeReader.**get_path_to_osm_file**(*subregion_name*, *osm_file_format*, *data_dir=None*)

Get the path to an OSM data file (if available) of a specific file format for a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on BBBike free download server
- **osm_file_format** (*str*) – format (file extension) of an OSM data
- **data_dir** (*str or None*) – directory where the data file is located/saved; if None (default), the default directory

Returns path to the data file

Return type str or None

Example:

```
>>> import os
>>> from pydriosm.reader import BBBikeReader

>>> bbbike_reader = BBBikeReader()

>>> region_name = 'Leeds'
>>> file_format = ".pbf"
>>> dat_dir = "tests"

>>> path_to_leeds_pbf = bbbike_reader.Downloader.download_osm_data(
...     region_name, file_format, dat_dir, verbose=True, ret_download_path=True)
To download .pbf data of the following geographic region(s):
    Leeds
? [No]|Yes: yes
Downloading "Leeds.osm.pbf" to "tests\" ... Done.

>>> path_to_leeds_pbf_ = bbbike_reader.get_path_to_osm_file(
...     region_name, file_format, dat_dir)
>>> print(os.path.relpath(path_to_leeds_pbf_))
```

(continues on next page)

(continued from previous page)

```
tests\Leeds.osm.pbf

>>> print(path_to_leeds_pbf == path_to_leeds_pbf_)
True

>>> # Delete the downloaded PBF data file
>>> os.remove(path_to_leeds_pbf_)
```

BBBikeReader.read_csv_xz

BBBikeReader.**read_csv_xz**(*subregion_name*, *data_dir=None*, *download_confirmation_required=True*, *verbose=False*)

Read a compressed CSV (.csv.xz) data file of a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on BBBike free download server
- **data_dir** (*str or None*) – directory where the .csv.xz data file is located/saved; if None (default), the default directory
- **download_confirmation_required** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Returns tabular data of the .csv.xz file

Return type pandas.DataFrame or None

Example:

```
>>> import os
>>> from pyhelpers.dir import cd
>>> from pydriosm.reader import BBBikeReader

>>> bbbike_reader = BBBikeReader()

>>> region_name = 'Leeds'
>>> dat_dir = "tests"

>>> leeds_csv = bbbike_reader.read_csv_xz(region_name, dat_dir, verbose=True)
To download .csv.xz data of the following geographic region(s):
    Leeds
? [No]|Yes: yes
Downloading "Leeds.osm.csv.xz" to "tests\" ... Done.
Parsing "tests\Leeds.osm.csv.xz" ... Done.

>>> leeds_csv.head()
   type      id feature
0  node  154915    None
1  node  154916    None
2  node  154921    None
3  node  154922    None
4  node  154923    None
```

(continues on next page)

(continued from previous page)

```
>>> # Delete the downloaded .csv.xz data file
>>> os.remove(cd(dat_dir, "Leeds.osm.csv.xz"))
```

BBBikeReader.read_geojson_xz

BBBikeReader.**read_geojson_xz**(*subregion_name*, *data_dir=None*, *fmt_geom=False*, *download_confirmation_required=True*, *verbose=False*)

Read a .geojson.xz data file of a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on BBBike free download server
- **data_dir** (*str or None*) – directory where the .geojson.xz data file is located/saved; if None (default), the default directory
- **fmt_geom** (*bool*) – whether to reformat coordinates into a geometric object, defaults to False
- **download_confirmation_required** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Returns tabular data of the .csv.xz file

Return type pandas.DataFrame or None

Examples:

```
>>> import os
>>> from pyhelpers.dir import cd
>>> from pydriosm.reader import BBBikeReader

>>> bbbike_reader = BBBikeReader()

>>> region_name = 'Leeds'
>>> dat_dir = "tests"

>>> leeds_geoj = bbbike_reader.read_geojson_xz(region_name, dat_dir, verbose=True)
To download .geojson.xz data of the following geographic region(s):
    Leeds
? [No]|Yes: yes
Downloading "Leeds.osm.geojson.xz" to "tests\" ... Done.
Parsing "tests\Leeds.osm.geojson.xz" ... Done.

>>> leeds_geoj.head()
  feature_name  ...                                          properties
0      Feature  ...  {'highway': 'motorway_junction', 'name': 'Flus...
1      Feature  ...  {'highway': 'motorway_junction', 'name': 'Bram...
2      Feature  ...  {'highway': 'motorway_junction', 'name': 'Bell...
3      Feature  ...  {'highway': 'motorway_junction', 'name': 'Loft...
4      Feature  ...  {'highway': 'motorway_junction', 'name': 'Loft...
[5 rows x 4 columns]
```

(continues on next page)

(continued from previous page)

```
>>> leeds_geoj[['coordinates']].head()
                coordinates
0  [-1.5558097, 53.6873431]
1     [-1.34293, 53.844618]
2   [-1.517335, 53.7499667]
3   [-1.514124, 53.7416937]
4   [-1.516511, 53.7256632]

>>> # Set `fmt_geom` to be True
>>> leeds_geoj_ = bbbike_reader.read_geojson_xz(region_name, dat_dir, fmt_
↪geom=True)

>>> leeds_geoj_[['coordinates']].head()
                     coordinates
0  POINT (-1.5558097 53.6873431)
1     POINT (-1.34293 53.844618)
2   POINT (-1.517335 53.7499667)
3   POINT (-1.514124 53.7416937)
4   POINT (-1.516511 53.7256632)

>>> # Delete the downloaded .csv.xz data file
>>> os.remove(cd(dat_dir, "Leeds.osm.geojson.xz"))
```

BBBikeReader.read_osm_pbf

BBBikeReader.**read_osm_pbf**(*subregion_name*, *data_dir=None*, *chunk_size_limit=50*, *parse_raw_feat=False*, *transform_geom=False*, *transform_other_tags=False*, *update=False*, *download_confirmation_required=True*, *pickle_it=False*, *ret_pickle_path=False*, *rm_osm_pbf=False*, *verbose=False*, ***kwargs*)

Read a PBF data file of a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on BBBike free download server
- **data_dir** (*str or None*) – directory where the PBF data file is saved; if None (default), the default directory
- **chunk_size_limit** (*int*) – threshold (in MB) that triggers the use of chunk parser, defaults to 50; if the size of the .osm.pbf file (in MB) is greater than chunk_size_limit, it will be parsed in a chunk-wise way
- **parse_raw_feat** (*bool*) – whether to parse each feature in the raw data, defaults to False
- **transform_geom** (*bool*) – whether to transform a single coordinate (or a collection of coordinates) into a geometric object, defaults to False
- **transform_other_tags** (*bool*) – whether to transform a 'other_tags' into a dictionary, defaults to False

- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to `False`
- **download_confirmation_required** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to `True`
- **pickle_it** (*bool*) – whether to save the .pbf data as a .pickle file, defaults to `False`
- **ret_pickle_path** (*bool*) – (when `pickle_it=True`) whether to return a path to the saved pickle file
- **rm_osm_pbf** (*bool*) – whether to delete the downloaded .osm.pbf file, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`
- **kwargs** – optional parameters of *parse_osm_pbf()*

Returns dictionary of the .osm.pbf data; when `pickle_it=True`, return a tuple of the dictionary and a path to the pickle file

Return type dict or tuple or None

Example:

```
>>> import os
>>> from pyhelpers.dir import cd
>>> from pydriosm.reader import BBBikeReader

>>> bbbike_reader = BBBikeReader()

>>> region_name = 'Leeds'
>>> dat_dir = "tests"

>>> # (Parsing the data in this example might take up to a few minutes.)
>>> leeds_osm_pbf = bbbike_reader.read_osm_pbf(region_name, dat_dir,
...                                            parse_raw_feat=True,
...                                            transform_geom=True,
...                                            transform_other_tags=True,
...                                            verbose=True)
To download .pbf data of the following geographic region(s):
    Leeds
? [No]|Yes: yes
Downloading "Leeds.osm.pbf" to "tests\" ... Done.
Parsing "tests\Leeds.osm.pbf" ... Done.

>>> list(leeds_osm_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> # Data of the 'multipolygons' layer
>>> leeds_osm_pbf_multipolygons = leeds_osm_pbf['multipolygons']

>>> leeds_osm_pbf_multipolygons.head()
      id                                        coordinates  ... tourism other_tags
0  10595  (POLYGON ((-1.5030223 53.6725382, -1.5034495 5...  ...    None       None
1  10600  (POLYGON ((-1.5116994 53.6764287, -1.5099361 5...  ...    None       None
2  10601  (POLYGON ((-1.5142403 53.6710831, -1.5143686 5...  ...    None       None
3  10612  (POLYGON ((-1.5129341 53.6704885, -1.5131883 5...  ...    None       None
4  10776  (POLYGON ((-1.5523801 53.7029081, -1.5522831 5...  ...    None       None
[5 rows x 27 columns]
```

(continues on next page)

(continued from previous page)

```
>>> # Delete the downloaded PBF data file
>>> os.remove(cd(dat_dir, "Leeds.osm.pbf"))
```

BBBikeReader.read_shp_zip

BBBikeReader.**read_shp_zip**(*subregion_name*, *layer_names=None*, *feature_names=None*, *data_dir=None*, *update=False*, *download_confirmation_required=True*, *pickle_it=False*, *ret_pickle_path=False*, *rm_extracts=False*, *rm_shp_zip=False*, *verbose=False*)

Read a shapefile of a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) that is available on BBBike free download server
- **layer_names** (*str or list or None*) – name of a .shp layer, e.g. 'railways', or names of multiple layers; if `None` (default), all available layers
- **feature_names** (*str or list or None*) – name of a feature, e.g. 'rail', or names of multiple features; if `None` (default), all available features
- **data_dir** (*str or None*) – directory where the .shp.zip data file is located/saved; if `None`, the default directory
- **update** (*bool*) – whether to check to update pickle backup (if available), defaults to `False`
- **download_confirmation_required** (*bool*) – whether to ask for confirmation before starting to download a file, defaults to `True`
- **pickle_it** (*bool*) – whether to save the .shp data as a .pickle file, defaults to `False`
- **ret_pickle_path** (*bool*) – (when `pickle_it=True`) whether to return a path to the saved pickle file
- **rm_extracts** (*bool*) – whether to delete extracted files from the .shp.zip file, defaults to `False`
- **rm_shp_zip** (*bool*) – whether to delete the downloaded .shp.zip file, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Returns dictionary of the shapefile data, with keys and values being layer names and tabular data (in the format of geopandas.GeoDataFrame), respectively; when `pickle_it=True`, return a tuple of the dictionary and a path to the pickle file

Return type dict or tuple or None

Examples:

```
>>> import os
>>> from pydriosm.reader import BBBikeReader

>>> bbbike_reader = BBBikeReader()

>>> region_name = 'Birmingham'
>>> dat_dir = "tests"

>>> birmingham_shp = bbbike_reader.read_shp_zip(region_name, data_dir=dat_dir,
...                                             verbose=True)
To download .shp.zip data of the following geographic region(s):
    Birmingham
? [No]|Yes: yes
Downloading "Birmingham.osm.shp.zip" to "tests\" ... Done.
Extracting "tests\Birmingham.osm.shp.zip" ...
to "tests\"
Done.
Parsing files at "tests\Birmingham-shp\shape\" ... Done.

>>> list(birmingham_shp.keys())
['buildings',
 'landuse',
 'natural',
 'places',
 'points',
 'pofw',
 'pois',
 'railways']

>>> # Data of 'railways' layer
>>> birmingham_railways_shp = birmingham_shp['railways']

>>> birmingham_railways_shp.head()
    osm_id  ... shape_type
0      740  ...          3
1     2148  ...          3
2  2950000  ...          3
3  3491845  ...          3
4  3981454  ...          3
[5 rows x 5 columns]

>>> # Read data of 'road' layer only from the original .shp.zip file
>>> # (and delete all extracts)

>>> layer_name = 'roads'
>>> feat_name = None

>>> birmingham_roads_shp = bbbike_reader.read_shp_zip(region_name, layer_name,
...                                                   feat_name, data_dir=dat_dir,
...                                                   rm_extracts=True,
...                                                   verbose=True)
Parsing "tests\Birmingham-shp\shape\roads.shp" ... Done.
Deleting the extracts "tests\Birmingham-shp\" ... Done.

>>> list(birmingham_roads_shp.keys())
['roads']

>>> birmingham_roads_shp['roads'].head()
   osm_id  ... shape_type
0      37  ...          3
```

(continues on next page)

(continued from previous page)

```
1      38  ...          3
2      41  ...          3
3      42  ...          3
4      45  ...          3
[5 rows x 9 columns]

>>> # Read data of multiple layers and features from the original .shp.zip file
>>> # (and delete all extracts)

>>> lyr_names = ['railways', 'waterways']
>>> feat_names = ['rail', 'canal']

>>> bham_rw_rc_shp = bbbike_reader.read_shp_zip(region_name, lyr_names, feat_names,
...                                             dat_dir, rm_extracts=True,
...                                             rm_shp_zip=True, verbose=True)
Extracting the following layer(s):
    'railways'
    'waterways'
from "tests\Birmingham.osm.shp.zip" ...
to "tests\"
Done.
Parsing files at "tests\Birmingham-shp\shape\" ... Done.
Deleting the extracts "tests\Birmingham-shp\" ... Done.
Deleting "tests\Birmingham.osm.shp.zip" ... Done.

>>> list(bham_rw_rc_shp.keys())
['railways', 'waterways']

>>> # Data of the 'railways' layer
>>> bham_rw_rc_shp_railways = bham_rw_rc_shp['railways']
>>> bham_rw_rc_shp_railways[['type', 'name']].head()
   type                                          name
0  rail                                Cross-City Line
1  rail                                Cross-City Line
2  rail  Derby to Birmingham (Proof House Junction) Line
3  rail                 Birmingham to Peterborough Line
4  rail          Water Orton to Park Lane Junction Curve

>>> # Data of the 'waterways' layer
>>> bham_rw_rc_shp_waterways = bham_rw_rc_shp['waterways']
>>> bham_rw_rc_shp_waterways[['type', 'name']].head()
     type                                          name
2   canal                      Birmingham and Fazeley Canal
8   canal                      Birmingham and Fazeley Canal
9   canal  Birmingham Old Line Canal Navigations - Rotton P
10  canal                               Oozells Street Loop
11  canal                      Worcester & Birmingham Canal
```

Attributes

DataDir

BBBikeReader.DataDir

property BBBikeReader.**DataDir**

Parsers for .osm.pbf / .osm.bz2 file

get_osm_pbf_layer_names(path_to_osm_pbf	Get indices and names of all layers in a PBF data file.
parse_osm_pbf_layer(pbf_layer_data, geo_typ, ...)	Parse data of a layer of PBF data.
parse_osm_pbf(path_to_osm_pbf[, ...])	Parse a PBF data file.

3.2.3 get_osm_pbf_layer_names

pydriosm.reader.**get_osm_pbf_layer_names**(*path_to_osm_pbf*)

Get indices and names of all layers in a PBF data file.

Parameters **path_to_osm_pbf** (*str*) – path to a PBF data file

Returns indices and names of each layer of the PBF data file

Return type dict

Example:

```
>>> import os
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pydriosm.reader import get_osm_pbf_layer_names

>>> # Download the PBF data file of Rutland as an example
>>> geofabrik_downloader = GeofabrikDownloader()

>>> path_to_rutland_pbf = geofabrik_downloader.download_osm_data(
...     subregion_names='Rutland', osm_file_format=".pbf", download_dir="tests",
...     verbose=True, ret_download_path=True)
To download .osm.pbf data of the following geographic region(s):
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "tests\" ... Done.

>>> # Get indices and names of all layers in the downloaded PBF data file
>>> lyr_idx_names = get_osm_pbf_layer_names(path_to_rutland_pbf)

>>> for k, v in lyr_idx_names.items():
...     print(f'{k}: {v}')
0: points
1: lines
2: multilinestrings
3: multipolygons
4: other_relations

>>> # Delete the downloaded PBF data file
>>> os.remove(path_to_rutland_pbf)
```

3.2.4 parse_osm_pbf_layer

pydriosm.reader.**parse_osm_pbf_layer**(*pbf_layer_data*, *geo_typ*, *transform_geom*, *transform_other_tags*)

Parse data of a layer of PBF data.

Parameters

- **pbf_layer_data** (*pandas.DataFrame*) – data of a specific layer of PBF data.
- **geo_typ** (*str*) – geometric type
- **transform_geom** (*bool*) – whether to transform a single coordinate (or a collection of coordinates) into a geometric object
- **transform_other_tags** (*bool*) – whether to transform a 'other_tags' into a dictionary

Returns parsed data of the geo_typ layer of a given .pbf file

Return type pandas.DataFrame

See the examples for the function *parse_osm_pbf()*.

3.2.5 parse_osm_pbf

pydriosm.reader.**parse_osm_pbf**(*path_to_osm_pbf*, *parse_raw_feat=False*, *transform_geom=False*, *transform_other_tags=False*, *number_of_chunks=None*, *max_tmpfile_size=None*)

Parse a PBF data file.

Parameters

- **path_to_osm_pbf** (*str*) – path to a PBF data file
- **parse_raw_feat** (*bool*) – whether to parse each feature in the raw data, defaults to False
- **transform_geom** – whether to transform a single coordinate (or a collection of coordinates) into a geometric object, defaults to False
- **transform_other_tags** (*bool*) – whether to transform a 'other_tags' into a dictionary, defaults to False
- **number_of_chunks** (*int or None*) – number of chunks, defaults to None
- **max_tmpfile_size** (*int or None*) – defaults to None; see also *gdal_configurations()*

Returns parsed OSM PBF data

Return type dict

Note: The driver categorises features into 5 layers:

- **0: 'points'** - "node" features having significant tags attached
- **1: 'lines'** - "way" features being recognized as non-area

- **2: 'multilinestrings'** - "relation" features forming a multilinestring (type='multilinestring' / type='route')
- **3: 'multipolygons'** - "relation" features forming a multipolygon (type='multipolygon' / type='boundary'), and "way" features being recognized as area
- **4: 'other_relations'** - "relation" features not belonging to the above 2 layers

See also [POP-1].

This function may require fairly high amount of physical memory to parse large files (e.g. > 200MB), in which case it would be recommended that `number_of_chunks` is set to be a reasonable value.

Example:

```
>>> import os
>>> from pydriosm.reader import GeofabrikDownloader, parse_osm_pbf

>>> # Download the PBF data file of Rutland as an example
>>> geofabrik_downloader = GeofabrikDownloader()

>>> path_to_rutland_pbf = geofabrik_downloader.download_osm_data(
...     subregion_names='Rutland', osm_file_format=".pbf", download_dir="tests",
...     verbose=True, ret_download_path=True)
To download .osm.pbf data of the following geographic region(s):
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "tests\" ... Done.

>>> print(os.path.relpath(path_to_rutland_pbf))
tests\rutland-latest.osm.pbf

>>> # Parse the downloaded PBF data
>>> rutland_pbf_raw = parse_osm_pbf(path_to_rutland_pbf)

>>> type(rutland_pbf_raw)
dict
>>> list(rutland_pbf_raw.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> rutland_pbf_raw_points = rutland_pbf_raw['points']
>>> rutland_pbf_raw_points.head()
                                              points
0  {"type": "Feature", "geometry": {"type": "Poin...
1  {"type": "Feature", "geometry": {"type": "Poin...
2  {"type": "Feature", "geometry": {"type": "Poin...
3  {"type": "Feature", "geometry": {"type": "Poin...
4  {"type": "Feature", "geometry": {"type": "Poin...

>>> # Set ``parse_raw_feat`` to be ``True``
>>> rutland_pbf_parsed_0 = parse_osm_pbf(path_to_rutland_pbf, parse_raw_feat=True)

>>> rutland_pbf_parsed_points_0 = rutland_pbf_parsed_0['points']
>>> rutland_pbf_parsed_points_0.head()
         id              coordinates  ... man_made                   other_tags
0    488432  [-0.5134241, 52.6555853]  ...     None              "odbl"=>"clean"
1    488658  [-0.5313354, 52.6737716]  ...     None                         None
2  13883868  [-0.7229332, 52.5889864]  ...     None                         None
```

(continues on next page)

(continued from previous page)

```
3  14049101  [-0.7249922, 52.6748223]  ...     None  "traffic_calming"=>"cushion"
4  14558402  [-0.7266686, 52.6695051]  ...     None      "direction"=>"clockwise"
[5 rows x 12 columns]

>>> # Set both ``parse_raw_feat`` and ``transform_geom`` to be ``True``
>>> rutland_pbf_parsed_1 = parse_osm_pbf(path_to_rutland_pbf, parse_raw_feat=True,
...                                      transform_geom=True)

>>> rutland_pbf_parsed_points_1 = rutland_pbf_parsed_1['points']
>>> # Check the difference in 'coordinates', compared to ``rutland_pbf_parsed_points_0``
>>> rutland_pbf_parsed_points_1[['coordinates']].head()
                              coordinates
0            POINT (-0.5134241 52.6555853)
1            POINT (-0.5313354 52.6737716)
2   POINT (-0.7229332000000001 52.5889864)
3            POINT (-0.7249922 52.6748223)
4            POINT (-0.7266686 52.6695051)

>>> # Further, set ``transform_other_tags`` to be ``True``
>>> rutland_pbf_parsed_2 = parse_osm_pbf(path_to_rutland_pbf, parse_raw_feat=True,
...                                      transform_other_tags=True)

>>> rutland_pbf_parsed_points_2 = rutland_pbf_parsed_2['points']
>>> # Check the difference in 'other_tags', compared to ``rutland_pbf_parsed_points_0``
>>> rutland_pbf_parsed_points_2[['other_tags']].head()
                      other_tags
0              {'odbl': 'clean'}
1                           None
2                           None
3  {'traffic_calming': 'cushion'}
4     {'direction': 'clockwise'}

>>> # Delete the downloaded PBF data file
>>> os.remove(path_to_rutland_pbf)
```

See also:

More examples for the method *GeofabrikReader.read_osm_pbf()*.

Parsers for .shp / .shp.zip file

unzip_shp_zip(path_to_shp_zip[, ...])	Unzip a .shp.zip file.
read_shp_file(path_to_shp[, method, emulate_gpd])	Parse a shapefile.
get_epsg4326_wgs84_crs_ref([as_str])	Get reference of EPSG Projection 4326 - WGS 84 (EPSG:4326) for the setting of CRS for saving shapefile data.
get_epsg4326_wgs84_prj_ref()	Get reference of EPSG Projection 4326 - WGS 84 (EPSG:4326) for saving shapefile projection data.
make_pyshp_fields(shp_data, field_names, ...)	Make fields data for writing shapefiles by pyshp.

continues on next page

Table 11 – continued from previous page	
write_to_shapefile(shp_data, path_to_shp[, ...])	Save .shp data as a shapefile by pyshp.
parse_layer_shp(path_to_layer_shp[, ...])	Parse a layer of OSM shapefile data.
merge_shps(paths_to_shp_files, ...[, method])	Merge multiple shapefiles.
merge_layer_shps(paths_to_shp_zip_files, ...)	Merge shapefiles over a layer for multiple geographic regions.

3.2.6 unzip_shp_zip

pydriosm.reader.**unzip_shp_zip**(*path_to_shp_zip*, *path_to_extract_dir=None*, *layer_names=None*, *mode='r'*, *clustered=False*, *verbose=False*, *ret_extract_dir=False*)

Unzip a .shp.zip file.

Parameters

- **path_to_shp_zip** (*str*) – path to a zipped shapefile data (.shp.zip)
- **path_to_extract_dir** (*str or None*) – path to a directory where extracted files will be saved; if `None` (default), the same directory where the .shp.zip file is saved
- **layer_names** (*str or list or None*) – name of a .shp layer, e.g. 'railways', or names of multiple layers; if `None` (default), all available layers
- **mode** (*str*) – the `mode` parameter of zipfile.ZipFile(), defaults to `'r'`
- **clustered** (*bool*) – whether to put the data files of different layer in respective folders, defaults to `False`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`
- **ret_extract_dir** (*bool*) – whether to return the path to the directory where extracted files are saved, defaults to `False`

Returns the path to the directory of extracted files when `ret_extract_dir` is set to be `True`

Return type str

Examples:

```
>>> import os
>>> from pyhelpers.dir import cd, delete_dir
>>> from pydriosm.reader import GeofabrikDownloader, unzip_shp_zip

>>> geofabrik_downloader = GeofabrikDownloader()

>>> path_to_rutland_shp_zip = geofabrik_downloader.download_osm_data(
...     subregion_names='Rutland', osm_file_format=".shp.zip", download_dir="tests",
...     verbose=True, ret_download_path=True)
To download .shp.zip data of the following geographic region(s):
```

(continues on next page)

(continued from previous page)

```
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest-free.shp.zip" to "tests\" ... Done.

>>> layer_name = 'railways'

>>> unzip_shp_zip(path_to_rutland_shp_zip, layer_names=layer_name, verbose=True)
Extracting the following layer(s):
    'railways'
from "tests\rutland-latest-free.shp.zip" ...
to "tests\rutland-latest-free-shp\"

>>> path_to_rutland_shp_dir = unzip_shp_zip(path_to_rutland_shp_zip, verbose=True,
...                                         ret_extract_dir=True)
Extracting "tests\rutland-latest-free.shp.zip" ...
to "tests\rutland-latest-free-shp\"
Done.

>>> print(os.path.relpath(path_to_rutland_shp_dir))
tests\rutland-latest-free-shp

>>> lyr_names = ['railways', 'transport', 'traffic']

>>> paths_to_layer_dirs = unzip_shp_zip(path_to_rutland_shp_zip, layer_names=lyr_names,
...                                     clustered=True, verbose=2, ret_extract_dir=True)
Extracting the following layer(s):
    'railways'
    'transport'
    'traffic'
from "tests\rutland-latest-free.shp.zip" ...
to "tests\rutland-latest-free-shp\"
Done.
Clustering layer data ...
    traffic ... Done.
    transport_a ... Done.
    transport ... Done.
    railways ... Done.
    traffic_a ... Done.
All done.

>>> for path_to_lyr_dir in paths_to_layer_dirs:
...     print(os.path.relpath(path_to_lyr_dir))
tests\rutland-latest-free-shp\railways
tests\rutland-latest-free-shp\transport
tests\rutland-latest-free-shp\traffic

>>> # Delete the extracted files
>>> delete_dir(os.path.commonpath(paths_to_layer_dirs), verbose=True)
The directory "tests\rutland-latest-free-shp\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "tests\rutland-latest-free-shp\" ... Done.

>>> # Delete the downloaded .shp.zip data file
>>> os.remove(path_to_rutland_shp_zip)
```

3.2.7 read_shp_file

pydriosm.reader.**read_shp_file**(*path_to_shp*, *method='pyshp'*, *emulate_gpd=False*, ***kwargs*)

Parse a shapefile.

Parameters

- **path_to_shp** – path to a .shp data file
- **method** (*str*) – method used to read shapefiles; options include: `'pyshp'` (default) and `'geopandas'` (or `'gpd'`) if `method='geopandas'` (or `method='gpd'`), this function relies on geopandas.read_file(); otherwise, it by default uses shapefile.Reader()
- **emulate_gpd** (*bool*) – whether to emulate the data format produced by geopandas.read_file(), when `method='pyshp'`.
- **kwargs** – optional parameters of geopandas.read_file() or shapefile.Reader()

Type str

Returns data frame of the .shp data

Return type pandas.DataFrame or geopandas.GeoDataFrame

Note: If `method` is set to be `'geopandas'` (or `'gpd'`), it requires availability of the package GeoPandas.

Examples:

```
>>> import os
>>> from pyhelpers.dir import cd, delete_dir
>>> from pydriosm.reader import GeofabrikDownloader, unzip_shp_zip, read_shp_file

>>> # Download the .shp.zip file of Rutland as an example
>>> geofabrik_downloader = GeofabrikDownloader()

>>> sr_name = 'Rutland'
>>> file_fmt = ".shp"
>>> dwnld_dir = "tests"

>>> path_to_rutland_shp_zip = geofabrik_downloader.download_osm_data(
...     sr_name, file_fmt, dwnld_dir, verbose=True, ret_download_path=True)
To download .shp.zip data of the following geographic region(s):
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest-free.shp.zip" to "tests\" ... Done.

>>> path_to_rutland_shp_dir = unzip_shp_zip(path_to_shp_zip=path_to_rutland_shp_zip,
...                                         ret_extract_dir=True)

>>> # .shp data of 'railways'
>>> railways_shp_filename = "gis_osm_railways_free_1.shp"
>>> path_to_rutland_railways_shp = cd(path_to_rutland_shp_dir, railways_shp_filename)

>>> # Set `method` to be 'gpd' or 'geopandas'
>>> rutland_railways_shp = read_shp_file(path_to_rutland_railways_shp)
```

(continues on next page)

(continued from previous page)

```
>>> rutland_railways_shp.head()
    osm_id  code  ...                                        coordinates shape_type
0  2162114  6101  ...  [(-0.4528083, 52.6993402), (-0.4518933, 52.698...          3
1  3681043  6101  ...  [(-0.6531215, 52.5730787), (-0.6531793, 52.572...          3
2  3693985  6101  ...  [(-0.7323403, 52.6782102), (-0.7319059, 52.678...          3
3  3693986  6101  ...  [(-0.6173072, 52.6132317), (-0.6241869, 52.614...          3
4  4806329  6101  ...  [(-0.4576926, 52.7035194), (-0.4565358, 52.702...          3
[5 rows x 9 columns]

>>> # Set `emulate_gpd` to be True
>>> rutland_railways_shp = read_shp_file(path_to_rutland_railways_shp, emulate_gpd=True)

>>> rutland_railways_shp.head()
    osm_id  code  ... tunnel                                           geometry
0  2162114  6101  ...      F  LINESTRING (-0.4528083 52.6993402, -0.4518933 ...
1  3681043  6101  ...      F  LINESTRING (-0.6531215 52.5730787, -0.6531793 ...
2  3693985  6101  ...      F  LINESTRING (-0.7323403000000001 52.6782102, -0...
3  3693986  6101  ...      F  LINESTRING (-0.6173071999999999 52.6132317, -0...
4  4806329  6101  ...      F  LINESTRING (-0.4576926 52.7035194, -0.4565358 ...
[5 rows x 8 columns]

>>> # Alternatively, set `method` to be 'gpd' to use GeoPandas
>>> rutland_railways_shp_ = read_shp_file(path_to_rutland_railways_shp, method='gpd')

>>> rutland_railways_shp_.head()
    osm_id  code  ... tunnel                                           geometry
0  2162114  6101  ...      F  LINESTRING (-0.45281 52.69934, -0.45189 52.698...
1  3681043  6101  ...      F  LINESTRING (-0.65312 52.57308, -0.65318 52.572...
2  3693985  6101  ...      F  LINESTRING (-0.73234 52.67821, -0.73191 52.678...
3  3693986  6101  ...      F  LINESTRING (-0.61731 52.61323, -0.62419 52.614...
4  4806329  6101  ...      F  LINESTRING (-0.45769 52.70352, -0.45654 52.702...
[5 rows x 8 columns]

>>> len(rutland_railways_shp) == len(rutland_railways_shp_)
True

>>> # Delete the extracted shapefiles
>>> delete_dir(path_to_rutland_shp_dir, verbose=True)
The directory "tests\rutland-latest-free-shp\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "tests\rutland-latest-free-shp\" ... Done.

>>> # Delete the downloaded shapefile
>>> os.remove(path_to_rutland_shp_zip)
```

3.2.8 get_epsg4326_wgs84_crs_ref

pydriosm.reader.**get_epsg4326_wgs84_crs_ref**(*as_str=False*)

Get reference of EPSG Projection 4326 - WGS 84 (EPSG:4326) for the setting of CRS for saving shapefile data.

Parameters **as_str** (*bool*) – whether to return the reference as a string type

Returns reference of EPSG Projection 4326 - WGS 84 (in Proj4 format)

Return type dict or str

Example:

```
>>> from pydriosm.reader import get_epsg4326_wgs84_crs_ref

>>> shp_crs = get_epsg4326_wgs84_crs_ref()
>>> print(shp_crs)
{'proj': 'longlat', 'ellps': 'WGS84', 'datum': 'WGS84', 'no_defs': True}

>>> shp_crs = get_epsg4326_wgs84_crs_ref(as_str=True)
>>> print(shp_crs)
+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
```

3.2.9 get_epsg4326_wgs84_prj_ref

pydriosm.reader.**get_epsg4326_wgs84_prj_ref**()

Get reference of EPSG Projection 4326 - WGS 84 (EPSG:4326) for saving shapefile projection data.

Returns reference of EPSG Projection 4326 - WGS 84 (in ESRI WKT format)

Return type str

Example:

```
>>> from pydriosm.reader import get_epsg4326_wgs84_prj_ref

>>> epsg4326_wgs84_prj_ref = get_epsg4326_wgs84_prj_ref()
>>> print(epsg4326_wgs84_prj_ref)
GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137,298.257223563]],...
```

See also:

Source: https://spatialreference.org/ref/epsg/4326/esriwkt/

3.2.10 make_pyshp_fields

pydriosm.reader.**make_pyshp_fields**(*shp_data*, *field_names*, *decimal_precision*)

Make fields data for writing shapefiles by pyshp.

Parameters

- **shp_data** (*pandas.DataFrame*) – .shp data
- **field_names** (*list or pandas.Index*) – names of fields to be written as shapefile records
- **decimal_precision** (*int*) – decimal precision for writing float records

Returns list of records in the .shp data

Return type list

3.2.11 write_to_shapefile

pydriosm.reader.**write_to_shapefile**(*shp_data*, *path_to_shp*, *decimal_precision=5*, *prj_file=True*)

Save .shp data as a shapefile by pyshp.

Parameters

- **shp_data** (*pandas.DataFrame*) – .shp data
- **path_to_shp** (*str*) – path where the .shp data is saved
- **decimal_precision** (*int*) – decimal precision for writing float records, defaults to 5
- **prj_file** (*bool*) – whether to create a .prj projection file for the shapefile, defaults to True

Example:

```
>>> import os
>>> import glob
>>> from pyhelpers.dir import cd, delete_dir
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pydriosm.reader import read_shp_file, unzip_shp_zip, write_to_shapefile

>>> # Download the .shp.zip file of Rutland as an example
>>> geofabrik_downloader = GeofabrikDownloader()

>>> sr_name = 'Rutland'

>>> path_to_rutland_shp_zip = geofabrik_downloader.download_osm_data(
...     sr_name, osm_file_format=".shp", download_dir="tests",
...     confirmation_required=False, ret_download_path=True)

>>> # Extract the downloaded .shp.zip file
>>> rutland_shp_dir = unzip_shp_zip(path_to_rutland_shp_zip, layer_names='railways',
...                                 ret_extract_dir=True)

>>> railways_shp_filename = glob.glob1(rutland_shp_dir, "*.shp")[0]
>>> path_to_railways_shp = cd(rutland_shp_dir, railways_shp_filename)

>>> # Read the .shp file
>>> rutland_railways_shp = read_shp_file(path_to_railways_shp)

>>> # Save the railways data as "tests\rutland\railways.shp"
>>> save_shp_path = "tests\rutland\railways"  # with or without the extension ".shp"
>>> write_to_shapefile(rutland_railways_shp, save_shp_path)

>>> # Read the saved the .shp file
>>> rutland_railways_shp_ = read_shp_file(save_shp_path)

>>> # Check if the retrieved .shp data is equal to the original one
>>> rutland_railways_shp_.equals(rutland_railways_shp)
True

>>> # Delete the extracted data files
>>> delete_dir(rutland_shp_dir, confirmation_required=False, verbose=True)
Deleting "tests\rutland-latest-free-shp\" ... Done.
>>> delete_dir("tests\rutland", confirmation_required=False, verbose=True)
Deleting "tests\rutland\" ... Done.
```

(continues on next page)

(continued from previous page)

```
>>> # Delete the downloaded shapefile
>>> os.remove(path_to_rutland_shp_zip)
```

3.2.12 parse_layer_shp

pydriosm.reader.**parse_layer_shp**(*path_to_layer_shp*, *feature_names=None*, *crs=None*, *save_fclass_shp=False*, *driver='ESRI Shapefile'*, *ret_path_to_fclass_shp=False*, ***kwargs*)

Parse a layer of OSM shapefile data.

Parameters

- **path_to_layer_shp** (*str or list*) – path(s) to one (or multiple) shapefile(s)
- **feature_names** (*str or list or None*) – class name(s) of feature(s), defaults to None
- **crs** (*dict*) – specification of coordinate reference system; if None (default), check specify_shp_crs()
- **save_fclass_shp** (*bool*) – (when fclass is not None) whether to save data of the fclass as shapefile, defaults to False
- **driver** (*str*) – the OGR format driver, defaults to 'ESRI Shapefile'; see also the driver parameter of geopandas.GeoDataFrame.to_file()
- **ret_path_to_fclass_shp** (*bool*) – (when save_fclass_shp is True) whether to return the path to the saved data of fclass, defaults to False
- **kwargs** – optional parameters of *read_shp_file()*

Returns parsed shapefile data

Return type geopandas.GeoDataFrame

Examples:

```
>>> import os
>>> from pyhelpers.dir import cd, delete_dir
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pydriosm.reader import parse_layer_shp, unzip_shp_zip

>>> # Download the .shp.zip file of Rutland as an example
>>> geofabrik_downloader = GeofabrikDownloader()

>>> sr_name = 'Rutland'

>>> path_to_rutland_shp_zip = geofabrik_downloader.download_osm_data(
...     sr_name, osm_file_format=".shp", download_dir="tests",
...     confirmation_required=False, ret_download_path=True)

>>> # Extract the downloaded .shp.zip file
>>> rutland_shp_dir = unzip_shp_zip(path_to_rutland_shp_zip, ret_extract_dir=True)
>>> path_to_railways_shp = cd(rutland_shp_dir, "gis_osm_railways_free_1.shp")

>>> # Parse the 'railways' layer
```

(continues on next page)

(continued from previous page)

```
>>> rutland_railways_shp = parse_layer_shp(path_to_railways_shp)

>>> rutland_railways_shp.head()
    osm_id  code  ...                                        coordinates shape_type
0  2162114  6101  ...  [(-0.4528083, 52.6993402), (-0.4518933, 52.698...          3
1  3681043  6101  ...  [(-0.6531215, 52.5730787), (-0.6531793, 52.572...          3
2  3693985  6101  ...  [(-0.7323403, 52.6782102), (-0.7319059, 52.678...          3
3  3693986  6101  ...  [(-0.6173072, 52.6132317), (-0.6241869, 52.614...          3
4  4806329  6101  ...  [(-0.4576926, 52.7035194), (-0.4565358, 52.702...          3
[5 rows x 9 columns]

>>> rutland_railways_rail, path_to_rutland_railways_rail = parse_layer_shp(
...     path_to_railways_shp, feature_names='rail', save_fclass_shp=True,
...     ret_path_to_fclass_shp=True)

>>> rutland_railways_rail.head()
    osm_id  code  ...                                        coordinates shape_type
0  2162114  6101  ...  [(-0.4528083, 52.6993402), (-0.4518933, 52.698...          3
1  3681043  6101  ...  [(-0.6531215, 52.5730787), (-0.6531793, 52.572...          3
2  3693985  6101  ...  [(-0.7323403, 52.6782102), (-0.7319059, 52.678...          3
3  3693986  6101  ...  [(-0.6173072, 52.6132317), (-0.6241869, 52.614...          3
4  4806329  6101  ...  [(-0.4576926, 52.7035194), (-0.4565358, 52.702...          3
[5 rows x 9 columns]

>>> print(os.path.relpath(path_to_rutland_railways_rail))
tests\rutland-latest-free-shp\railways\gis_osm_railways_free_1_rail.shp

>>> # Delete the extracted data files
>>> delete_dir(rutland_shp_dir, verbose=True)
The directory "tests\rutland-latest-free-shp\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "tests\rutland-latest-free-shp\" ... Done.

>>> # Delete the downloaded shapefile
>>> os.remove(path_to_rutland_shp_zip)
```

3.2.13 merge_shps

pydriosm.reader.**merge_shps**(*paths_to_shp_files*, *path_to_merged_dir*, *method='pyshp'*)

Merge multiple shapefiles.

Parameters

- **paths_to_shp_files** (*list*) – list of paths to shapefiles (in .shp format)
- **path_to_merged_dir** (*str*) – path to a directory where the merged files are to be saved
- **method** (*str*) – the method used to merge/save shapefiles; options include: `'pyshp'` (default) and `'geopandas'` (or `'gpd'`) if `method='geopandas'`, this function relies on geopandas.GeoDataFrame.to_file(); otherwise, it by default uses shapefile.Writer()

Note: If `method` is set to be `'geopandas'` (or `'gpd'`), it requires availability of the

package GeoPandas.

See also:

- The example for the function *merge_layer_shps()*.
- Resource: https://github.com/GeospatialPython/pyshp

3.2.14 merge_layer_shps

pydriosm.reader.**merge_layer_shps**(*paths_to_shp_zip_files*, *layer_name*, *method='pyshp'*, *rm_zip_extracts=True*, *merged_shp_dir=None*, *rm_shp_temp=True*, *verbose=False*, *ret_merged_shp_path=False*)

Merge shapefiles over a layer for multiple geographic regions.

Parameters

- **paths_to_shp_zip_files** (*list*) – list of paths to data of shapefiles (in .shp.zip format)
- **layer_name** (*str*) – name of a layer (e.g. 'railways')
- **method** (*str*) – the method used to merge/save shapefiles; options include: `'pyshp'` (default) and `'geopandas'` (or `'gpd'`) if `method='geopandas'`, this function relies on geopandas.GeoDataFrame.to_file(); otherwise, it by default uses shapefile.Writer()
- **rm_zip_extracts** (*bool*) – whether to delete the extracted files, defaults to `False`
- **rm_shp_temp** (*bool*) – whether to delete temporary layer files, defaults to `False`
- **merged_shp_dir** (*str or None*) – if `None` (default), use the layer name as the name of the folder where the merged .shp files will be saved
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`
- **ret_merged_shp_path** (*bool*) – whether to return the path to the merged .shp file, defaults to `False`

Returns the path to the merged file when `ret_merged_shp_path=True`

Return type list or str

Note: This function does not create projection (.prj) for the merged map (see also [MMS-1])

For valid `layer_name`, check *get_valid_shp_layer_names()*.

Example:

```
>>> import os
>>> from pyhelpers.dir import delete_dir
>>> from pydriosm.downloader import GeofabrikDownloader
>>> from pydriosm.reader import merge_layer_shps, read_shp_file

>>> # To merge 'railways' layers of Greater Manchester and West Yorkshire"

>>> geofabrik_downloader = GeofabrikDownloader()

>>> sr_names = ['Greater Manchester', 'West Yorkshire']
>>> dat_dir = "tests"

>>> shp_zip_file_paths = geofabrik_downloader.download_osm_data(
...     sr_names, osm_file_format=".shp", download_dir=dat_dir,
...     confirmation_required=False, ret_download_path=True, verbose=True)
Downloading "greater-manchester-latest-free.shp.zip" to "tests\" ... Done.
Downloading "west-yorkshire-latest-free.shp.zip" to "tests\" ... Done.

>>> lyr_name = 'railways'

>>> merged_shp_path = merge_layer_shps(shp_zip_file_paths, layer_name=lyr_name,
...                                    verbose=True, ret_merged_shp_path=True)
Extracting the following layer(s):
    'railways'
from "tests\greater-manchester-latest-free.shp.zip" ...
to "tests\greater-manchester-latest-free-shp\"
Done.
Extracting the following layer(s):
    'railways'
from "tests\west-yorkshire-latest-free.shp.zip" ...
to "tests\west-yorkshire-latest-free-shp\"
Done.
Merging the following shapefiles:
    "greater-manchester_gis_osm_railways_free_1.shp"
    "west-yorkshire_gis_osm_railways_free_1.shp"
In progress ... Done.
Find the merged shapefile at "tests\greater-manchester_west-yorkshire_railways\".

>>> print(os.path.relpath(merged_shp_path))
tests\greater-manchester_west-yorksh...\greater-manchester_west-yorkshire_railways.shp

>>> # Read the merged .shp file
>>> merged_shp_data = read_shp_file(merged_shp_path)

>>> # Delete the merged shapefile
>>> delete_dir(os.path.dirname(merged_shp_path), verbose=True)
The directory "tests\greater-manchester_west-yorkshire_railways\" is not empty.
Confirmed to delete it? [No]|Yes: yes
Deleting "tests\greater-manchester_west-yorkshire_railways\" ... Done.

>>> # Delete the downloaded shapefiles
>>> for shp_zip_file_path in shp_zip_file_paths:
...     os.remove(shp_zip_file_path)
```

See also:

The examples for the method *GeofabrikReader.merge_subregion_layer_shp()*.

Parsers for .xz file

parse_csv_xz(path_to_csv_xz[, col_names])	Parse a compressed CSV (.csv.xz) data file.
parse_geojson_xz(path_to_geojson_xz[, fmt_geom])	Parse a compressed Osmium GeoJSON (.geojson.xz) data file.

3.2.15 parse_csv_xz

pydriosm.reader.**parse_csv_xz**(*path_to_csv_xz*, *col_names=None*)

Parse a compressed CSV (.csv.xz) data file.

Parameters

- **path_to_csv_xz** (*str*) – path to a .csv.xz data file
- **col_names** (*list or None*) – column names of .csv.xz data, defaults to None

Returns tabular data of the CSV file

Return type pandas.DataFrame

See the example for the method *BBBikeReader.read_csv_xz()*.

3.2.16 parse_geojson_xz

pydriosm.reader.**parse_geojson_xz**(*path_to_geojson_xz*, *fmt_geom=False*)

Parse a compressed Osmium GeoJSON (.geojson.xz) data file.

Parameters

- **path_to_geojson_xz** (*str*) – path to a .geojson.xz data file
- **fmt_geom** (*bool*) – whether to reformat coordinates into a geometric object, defaults to False

Returns tabular data of the Osmium GeoJSON file

Return type pandas.DataFrame

See the example for the method *BBBikeReader.read_geojson_xz()*.

3.3 ios

I/O and storage of OSM data extracts with PostgreSQL.

Classes

PostgresOSM([host, port, username, ...])	I/O and storage of OSM data extracts with PostgreSQL.

3.3.1 PostgresOSM

class pydriosm.ios.**PostgresOSM**(*host='localhost', port=5432, username='postgres', password=None, database_name='postgres', data_source='Geofabrik', data_dir=None, max_tmpfile_size=5000, **kwargs*)

I/O and storage of OSM data extracts with PostgreSQL.

Parameters

- **host** (*str or None*) – host address, defaults to `'localhost'` (or `'127.0.0.1'`)
- **port** (*int or None*) – port, defaults to `5432`
- **username** (*str or None*) – database username, defaults to `'postgres'`
- **password** (*str or int or None*) – database password, defaults to `None`
- **database_name** (*str*) – database name, defaults to `'postgres'`
- **data_source** (*str*) – name of data sources; options include `'Geofabrik'` (default) and `'BBBike'`
- **kwargs** – optional parameters of the class pyhelpers.sql.PostgreSQL

Variables

- **ValidDataSources** (*tuple*) – valid names of data sources
- **Downloaders** (*dict*) – instances of the classes for downloading data, including *GeofabrikDownloader* and *BBBikeDownloader*
- **Readers** (*dict*) – instances of the classes for downloading data, including *GeofabrikReader* and *BBBikeReader*
- **DataSource** (*str*) – name of data sources, options include `'Geofabrik'` and `'BBBike'`

Example:

```
>>> from pydriosm.ios import PostgresOSM

>>> database_name = 'osmdb_test'

>>> osmdb_test = PostgresOSM(database_name=database_name)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
```

(continues on next page)

(continued from previous page)

```
>>> print(osmdb_test.DataSource)
Geofabrik
>>> type(osmdb_test.Downloader)
pydriosm.downloader.GeofabrikDownloader
>>> type(osmdb_test.Reader)
pydriosm.reader.GeofabrikReader

>>> # Change the data source:
>>> osmdb_test.DataSource = 'BBBike'
>>> type(osmdb_test.Downloader)
pydriosm.downloader.BBBikeDownloader
>>> type(osmdb_test.Reader)
pydriosm.reader.BBBikeReader
```

Methods

alter_table_schema(table_name, schema_name, ...)	Move a table from one schema to another within the database being connected.
connect_database([database_name, verbose])	Establish a connection to a database of the PostgreSQL server being connected.
create_database(database_name[, verbose])	An alternative to sqlalchemy_utils.create_database.
create_schema(schema_name[, verbose])	Create a new schema in the database being connected.
create_table(table_name, column_specs[, ...])	Create a new table for the database being connected.
database_exists([database_name])	Check if a database exists in the PostgreSQL server being connected.
disconnect_database([database_name, verbose])	Kill the connection to a database in the PostgreSQL server being connected.
disconnect_other_databases()	Kill connections to all other databases of the PostgreSQL server being connected.
drop_database([database_name, ...])	Drop a database from the PostgreSQL server being connected.
drop_schema(schema_names[, ...])	Drop a schema in the database being connected.
drop_subregion_table(subregion_table_n	Delete all or specific schemas/layers of subregion data from the database being connected.
drop_table(table_name[, schema_name, ...])	Remove a table from the database being connected.
fetch_osm_data(subregion_name[, ...])	Fetch OSM data (of one or multiple layers) of a geographic region.
get_column_info(table_name[, schema_name, ...])	Get information about columns of a table.
get_database_size([database_name])	Get the size of a database in the PostgreSQL server being connected.

continues on next page

Table 14 – continued from previous page	
get_subregion_table_column_info(... [, ...])	Get information about columns of a specific schema and table data of a geographic region.
get_table_name_for_subregion(subregic	Get the default table name for a specific geographic region.
import_data(data, table_name[, schema_name, ...])	Import tabular data into the database being connected.
import_osm_data(osm_data, table_name[, ...])	Import OSM data into a database.
import_osm_layer(osm_layer_data, table_name, ...)	Import one layer of OSM data into a table.
import_subregion_osm_pbf(subregion_na ...])	Import data of geographic region(s) that do not have (sub-)subregions into a database.
list_table_names([schema_name, verbose])	List the names of all tables in a schema.
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 of the database being connected.
schema_exists(schema_name)	Check if a schema exists in the PostgreSQL server being connected.
subregion_table_exists(subregion_nam ...)	Check if a table (for a geographic region) exists.
table_exists(table_name[, schema_name])	Check if a table exists in the database being connected.

PostgresOSM.alter_table_schema

PostgresOSM.**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 database being connected.

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 as the function runs, defaults to `False`

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
 ↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> # Create a new table named "test_table" in the schema "testdb"
>>> new_table_name = 'test_table'
>>> column_specs = 'col_name_1 INT, col_name_2 TEXT'
>>> testdb.create_table(new_table_name, column_specs, verbose=True)
Creating a table: "public"."test_table" ... Done.

>>> # Create a new schema "test_schema"
>>> testdb.create_schema(schema_name='test_schema', verbose=True)
Creating a schema: "test_schema" ... Done.

>>> # Move the table "public"."test_table" to the schema "test_schema"
>>> testdb.alter_table_schema('test_table', 'public', '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.list_table_names(schema_name='test_schema')
>>> print(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.
```

PostgresOSM.connect_database

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

Establish a connection to a database of the PostgreSQL server being connected.

Parameters

- **database_name** (*str or None*) – name of a database; if 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

Example:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
 ↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> testdb.connect_database()
>>> print(testdb.database_name)
testdb
```

(continues on next page)

(continued from previous page)

```
>>> testdb.connect_database('postgres', verbose=True)
Connecting postgres:***@localhost:5432/postgres ... Successfully.

>>> print(testdb.database_name)
postgres
```

PostgresOSM.create_database

PostgresOSM.**create_database**(*database_name*, *verbose=False*)

An alternative to sqlalchemy_utils.create_database.

Parameters

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

Example:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> testdb.create_database('testdb1', verbose=True)
Creating a database: "testdb1" ... Done.

>>> print(testdb.database_name)
testdb1

>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb1" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb1" ... Done.
```

PostgresOSM.create_schema

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

Create a new schema in the database being connected.

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

Example:

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

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

(continues on next page)

(continued from previous page)

```
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> test_schema_name = 'test_schema'

>>> testdb.create_schema(test_schema_name, verbose=True)
Creating a schema: "test_schema" ... Done.

>>> testdb.schema_exists(test_schema_name)
True

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

PostgresOSM.create_table

PostgresOSM.**create_table**(*table_name*, *column_specs*, *schema_name='public'*, *verbose=False*)

Create a new table for the database being connected.

Parameters

- **table_name** (*str*) – name of a table
- **column_specs** (*str*) – specifications for each column of the table
- **schema_name** (*str*) – name of a schema, defaults to `'public'`
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to `False`

Example:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> tbl_name = 'test_table'
>>> column_specifications = 'col_name_1 INT, col_name_2 TEXT'

>>> testdb.create_table(tbl_name, column_specifications, verbose=True)
Creating a table "public"."test_table" ... Done.

>>> testdb.table_exists(tbl_name)
True

>>> test_tbl_col_info = testdb.get_column_info(tbl_name, as_dict=False)
```

(continues on next page)

(continued from previous page)

```
>>> 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
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

>>> # Drop the table
>>> testdb.drop_table(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.
```

PostgresOSM.database_exists

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

Check if a database exists in the PostgreSQL server being connected.

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

Returns True if the database exists, False, otherwise

Return type bool

Example:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
 ↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> testdb.database_exists('testdb')
True

>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.

>>> testdb.database_exists('testdb')
False
```

PostgresOSM.disconnect_database

PostgresOSM.**disconnect_database**(*database_name=None*, *verbose=False*)

Kill the connection to a database in the PostgreSQL server being connected.

If database_name is None (default), disconnect the current database.

Parameters

- **database_name** (*str or None*) – name of database to disconnect from, defaults to None
- **verbose** (*bool or int*) – whether to print relevant information in console as the function runs, defaults to False

Example:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
 ↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> print(testdb.database_name)
testdb
```

(continues on next page)

(continued from previous page)

```
>>> testdb.disconnect_database()
>>> print(testdb.database_name)
postgres
```

PostgresOSM.disconnect_other_databases

PostgresOSM.**disconnect_other_databases**()

Kill connections to all other databases of the PostgreSQL server being connected.

Example:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
 ↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> print(testdb.database_name)
testdb

>>> testdb.disconnect_other_databases()
>>> print(testdb.database_name)
postgres
```

PostgresOSM.drop_database

PostgresOSM.**drop_database**(*database_name=None*, *confirmation_required=True*, *verbose=False*)

Drop a database from the PostgreSQL server being connected.

If `database_name` is `None` (default), drop the current database.

Parameters

- **database_name** (*str or None*) – database to be disconnected, defaults to `None`
- **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`

Example:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
 ↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> # 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.

>>> testdb.database_exists(database_name='testdb')
False

>>> testdb.drop_database(database_name='testdb', verbose=True)
The database "testdb" does not exist.

>>> print(testdb.database_name)
postgres
```

PostgresOSM.drop_schema

PostgresOSM.**drop_schema**(*schema_names*, *confirmation_required=True*, *verbose=False*)

Drop a schema in the database being connected.

Parameters

- **schema_names** (*str or typing.Iterable*) – 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

Example:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> new_schema_names = ['points', 'lines', 'polygons']
>>> new_schema_names_ = ['test_schema']

>>> 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.

>>> 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)

```
>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

PostgresOSM.drop_subregion_table

PostgresOSM.**drop_subregion_table**(*subregion_table_names*, *schema_names=None*, *table_named_as_subregion=False*, *schema_named_as_layer=False*, *confirmation_required=True*, *verbose=False*)

Delete all or specific schemas/layers of subregion data from the database being connected.

Parameters

- **subregion_table_names** (*str or list*) – name of table for a subregion (or name of a subregion)
- **schema_names** (*list or None*) – names of schemas for each layer of the PBF data, if None (default), the default layer names as schema names
- **table_named_as_subregion** (*bool*) – whether to use subregion name as a table name, defaults to False
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name, defaults to False
- **confirmation_required** (*bool*) – whether to ask for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

Examples:

```
>>> from pydriosm.ios import PostgresOSM

>>> osmdb_test = PostgresOSM(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> # -- Import example data into the database ----------------------------------

>>> dat_dir = "tests"  # Specify a temporary data directory

>>> # Import PBF data of 'Rutland' (from Geofabrik free download server)
>>> osmdb_test.import_subregion_osm_pbf(
...     subregion_names='Rutland', data_dir=dat_dir, rm_osm_pbf=True,
...     confirmation_required=False)

>>> # (from BBBike free download server)
>>> osmdb_test.DataSource = 'BBBike'
```

(continues on next page)

(continued from previous page)

```
>>> # Import PBF data of 'Victoria' and 'Waterloo'
>>> osmdb_test.import_subregion_osm_pbf(
...     subregion_names=['Victoria', 'Waterloo'], data_dir=dat_dir,
...     parse_raw_feat=True, transform_geom=True, transform_other_tags=True,
...     rm_osm_pbf=True, confirmation_required=False)

>>> # Import shapefile data of 'Leeds' (from BBBike free download server)
>>> leeds_shp = osmdb_test.Reader.read_shp_zip(
...     subregion_name='Leeds', data_dir=dat_dir, rm_extracts=True,
...     rm_shp_zip=True, download_confirmation_required=False)

>>> osmdb_test.import_osm_data(leeds_shp, 'Leeds', confirmation_required=False)

>>> # -- Delete all data of Rutland and Leeds ----------------------------------
>>> subregion_tbl_names = ['Rutland', 'Leeds']

>>> osmdb_test.drop_subregion_table(subregion_tbl_names, verbose=True)
To drop tables from postgres:***@localhost:5432/osmdb_test:
    "Leeds"
    "Rutland"
 under the schemas:
    "buildings"
    "landuse"
    "lines"
    "multilinestrings"
    "multipolygons"
    "natural"
    "other_relations"
    "places"
    "points"
    "railways"
    "roads"
    "waterways"
? [No]|Yes: yes
Dropping the tables ...
    "buildings"."Leeds" ... Done.
    "landuse"."Leeds" ... Done.
    "lines"."Rutland" ... Done.
    "multilinestrings"."Rutland" ... Done.
    "multipolygons"."Rutland" ... Done.
    "natural"."Leeds" ... Done.
    "other_relations"."Rutland" ... Done.
    "places"."Leeds" ... Done.
    "points"."Leeds" ... Done.
    "points"."Rutland" ... Done.
    "railways"."Leeds" ... Done.
    "roads"."Leeds" ... Done.
    "waterways"."Leeds" ... Done.

>>> # -- Delete 'points' and 'other_relations' of Waterloo and Victoria ---------

>>> subregion_tbl_names = ['Waterloo', 'Victoria']
>>> lyr_schema_names = ['points', 'other_relations']

>>> osmdb_test.drop_subregion_table(subregion_tbl_names, lyr_schema_names,
...                                 verbose=True)
To drop tables from postgres:***@localhost:5432/osmdb_test:
    "Victoria"
    "Waterloo"
 under the schemas:
```

(continues on next page)

(continued from previous page)

```
    "other_relations"
    "points"
? [No]|Yes: yes
Dropping the tables ...
    "other_relations"."Victoria" ... Done.
    "other_relations"."Waterloo" ... Done.
    "points"."Victoria" ... Done.
    "points"."Waterloo" ... Done.

>>> # Delete the database 'osmdb_test'
>>> osmdb_test.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.drop_table

PostgresOSM.**drop_table**(*table_name*, *schema_name='public'*, *confirmation_required=True*, *verbose=False*)

Remove a table from the database being connected.

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str*) – name of a schema, defaults to `'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 as the function runs, defaults to `False`

See the example for the method `.create_table()`.

PostgresOSM.fetch_osm_data

PostgresOSM.**fetch_osm_data**(*subregion_name*, *layer_names=None*, *table_named_as_subregion=False*, *schema_named_as_layer=False*, *chunk_size=None*, *method='tempfile'*, *max_size_spooled=1*, *decode_geojson=False*, *decode_wkt=False*, *decode_other_tags=False*, *parse_geojson=False*, *sort_by='id'*, ***kwargs*)

Fetch OSM data (of one or multiple layers) of a geographic region.

See also [ROP-1]

Parameters

- **subregion_name** (*str*) – name of a geographic region (or the corresponding table)
- **layer_names** (*list or None*) – names of schemas for each layer of the PBF data, if `None` (default), the default layer names as schema names

- **table_named_as_subregion** (*bool*) – whether to use subregion name as a table name, defaults to `False`
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name, defaults to `False`
- **chunk_size** (*int or None*) – the number of rows in each batch to be written at a time, defaults to `None`
- **method** (*str or None*) – method to be used for buffering temporary data, defaults to `'tempfile'`
- **max_size_spooled** (*int , float*) – see pyhelpers.sql.PostgreSQL.read_sql_query, defaults to `1` (in GB)
- **decode_geojson** (*bool*) – whether to decode textual GeoJSON, defaults to `False`
- **decode_wkt** (*bool*) – whether to decode `'coordinates'` (if it is wkt), defaults to `False`
- **decode_other_tags** (*bool*) – whether to decode `'other_tags'` (if available), defaults to `False`
- **parse_geojson** (*bool*) – whether to parse raw GeoJSON (as it is raw feature data), defaults to `False`
- **sort_by** (*str or list*) – column name(s) by which the data (fetched from PostgreSQL) is sorted, defaults to `None`

Returns PBF (.osm.pbf) data

Return type dict

Example:

```
>>> from pydriosm.ios import PostgresOSM

>>> osmdb_test = PostgresOSM(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> # Import example data into the database:

>>> sr_name = 'Rutland'
>>> dat_dir = "tests"  # a temporary data directory

>>> osmdb_test.import_subregion_osm_pbf(
...     subregion_names=sr_name, data_dir=dat_dir, rm_osm_pbf=True,
...     confirmation_required=False)

>>> rutland_shp = osmdb_test.Reader.read_shp_zip(
...     subregion_name=sr_name, data_dir=dat_dir, rm_extracts=True,
...     rm_shp_zip=True, download_confirmation_required=False)
>>> osmdb_test.import_osm_data(rutland_shp, sr_name, confirmation_required=False)

>>> # Fetch data of all available layers from the database
>>> rutland_pbf = osmdb_test.fetch_osm_data(sr_name, table_named_as_subregion=True)

>>> type(rutland_pbf)
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(rutland_pbf.keys())
['points',
 'lines',
 'multilinestrings',
 'multipolygons',
 'other_relations',
 'buildings',
 'landuse',
 'natural',
 'places',
 'pofw',
 'pois',
 'railways',
 'roads',
 'traffic',
 'transport',
 'water',
 'waterways']

>>> # Fetch data of specific layers

>>> lyr_names = ['points', 'multipolygons']

>>> rutland_pbf_ = osmdb_test.fetch_osm_data(sr_name, lyr_names, sort_by='id')

>>> type(rutland_pbf_)
dict
>>> list(rutland_pbf_.keys())
# ['points', 'multipolygons']

>>> # Data of the 'points' layer
>>> rutland_pbf_points = rutland_pbf_['points']
>>> rutland_pbf_points.head()
                                              points
0  {"type": "Feature", "geometry": {"type": "Poin...
1  {"type": "Feature", "geometry": {"type": "Poin...
2  {"type": "Feature", "geometry": {"type": "Poin...
3  {"type": "Feature", "geometry": {"type": "Poin...
4  {"type": "Feature", "geometry": {"type": "Poin...

>>> # Parsed data
>>> rutland_pbf_parsed_ = osmdb_test.fetch_osm_data(
...     sr_name, lyr_names, decode_geojson=True, decode_wkt=True,
...     decode_other_tags=True, sort_by='id')

>>> # Parsed data of the 'points' layer
>>> rutland_pbf_parsed_points = rutland_pbf_parsed_['points']
>>> rutland_pbf_parsed_points.head()
         id  ...                       other_tags
0    488432  ...               {'odbl': 'clean'}
1    488658  ...                             None
2  13883868  ...                             None
3  14049101  ...  {'traffic_calming': 'cushion'}
4  14558402  ...      {'direction': 'clockwise'}
[5 rows x 12 columns]

>>> # Delete the database 'osmdb_test'
>>> osmdb_test.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

See also:

- More details of the above data can be found in the examples for the methods `.import_osm_data()` and `.import_subregion_osm_pbf()`.
- Similar examples about *fetching data from the database* are available in *Quick start*.

PostgresOSM.get_column_info

PostgresOSM.**get_column_info**(*table_name*, *schema_name='public'*, *as_dict=True*)

Get information about columns of a table.

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str*) – name of a schema, defaults to `'public'`
- **as_dict** (*bool*) – whether to return the column information as a dictionary, defaults to `True`

Returns information about each column of the given table

Return type pandas.DataFrame or dict

See the example for the method `.create_table()`.

PostgresOSM.get_database_size

PostgresOSM.**get_database_size**(*database_name=None*)

Get the size of a database in the PostgreSQL server being connected.

When `database_name` is `None` (default), the connected database is checked.

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

Returns size of the database

Return type int

Example:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> print(testdb.get_database_size())
7901 kB
```

PostgresOSM.get_subregion_table_column_info

PostgresOSM.**get_subregion_table_column_info**(*subregion_name*, *layer_name*, *as_dict=False*, *table_named_as_subregion=False*, *schema_named_as_layer=False*)

Get information about columns of a specific schema and table data of a geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region, which acts as a table name
- **layer_name** (*str*) – name of an OSM layer (e.g. 'points', 'railways', ...), which acts as a schema name
- **as_dict** (*bool*) – whether to return the column information as a dictionary, defaults to `True`
- **table_named_as_subregion** (*bool*) – whether to use subregion name as table name, defaults to `False`
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name, defaults to `False`

Returns information about each column of the given table

Return type pandas.DataFrame or dict

Examples:

```
>>> from pydriosm.ios import PostgresOSM

>>> osmdb_test = PostgresOSM(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> region_name = 'London'
>>> lyr_name = 'points'

>>> # Take for example a table named "points"."London"
>>> tbl_col_info = osmdb_test.get_subregion_table_column_info(region_name, lyr_
 ↪name)

>>> type(tbl_col_info)
pandas.core.frame.DataFrame
>>> tbl_col_info.index.to_list()[:5]
['table_catalog',
 'table_schema',
 'table_name',
 'column_name',
 'ordinal_position']

>>> # Another example of a table named "points"."Greater London"
>>> tbl_col_info_dict = osmdb_test.get_subregion_table_column_info(
...     region_name, lyr_name, as_dict=True, table_named_as_subregion=True,
...     schema_named_as_layer=True)

>>> type(tbl_col_info_dict)
dict
```

(continues on next page)

(continued from previous page)

```
>>> list(tbl_col_info_dict.keys())[:5]
['table_catalog',
 'table_schema',
 'table_name',
 'column_name',
 'ordinal_position']
```

PostgresOSM.get_table_name_for_subregion

PostgresOSM.**get_table_name_for_subregion**(*subregion_name*, *table_named_as_subregion=False*)

Get the default table name for a specific geographic region.

Parameters

- **subregion_name** (*str*) – name of a geographic region, which acts as a table name
- **table_named_as_subregion** (*bool*) – whether to use subregion name as table name, defaults to `False`

Returns default table name for storing the subregion data into the database

Return type str

Examples:

```
>>> from pydriosm.ios import PostgresOSM

>>> osmdb_test = PostgresOSM(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> region_name = 'London'

>>> tbl_name = osmdb_test.get_table_name_for_subregion(region_name)

>>> print(tbl_name)
London

>>> use_region_name = True
>>> tbl_name = osmdb_test.get_table_name_for_subregion(region_name, use_region_
 ↪name)

>>> print(tbl_name)
Greater London
```

Note: In the examples above, the default data source is 'Geofabrik'. Changing it to 'BBBike', the function may possibly produce a different output for the same input, as a geographic region that is included in one data source may not always be available from the other.

PostgresOSM.import_data

PostgresOSM.**import_data**(*data*, *table_name*, *schema_name='public'*, *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 the database being connected.

See also [SQL-P-DD-1] and [SQL-P-DD-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, defaults to `'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 to replace 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 typing.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 the example for the method `.read_sql_query()`.

PostgresOSM.import_osm_data

PostgresOSM.**import_osm_data**(*osm_data*, *table_name*, *schema_names=None*, *table_named_as_subregion=False*, *schema_named_as_layer=False*, *if_exists='replace'*, *force_replace=False*, *chunk_size=None*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Import OSM data into a database.

Parameters

- **osm_data** (*dict*) – OSM data of a geographic region
- **table_name** (*str*) – name of a table
- **schema_names** (*list or dict or None*) – names of schemas for each layer of the PBF data, if `None` (default), the default layer names as schema names
- **table_named_as_subregion** (*bool*) – whether to use subregion name as a table name, defaults to `False`
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name, defaults to `False`
- **if_exists** (*str*) – if the table already exists, to `'replace'` (default), `'append'` or `'fail'`
- **force_replace** (*bool*) – whether to force to replace 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`
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`
- **kwargs** – optional parameters of `.import_osm_pbf_layer()`

Examples:

```
>>> import os
>>> from pyhelpers.dir import cd
>>> from pydriosm.ios import PostgresOSM

>>> osmdb_test = PostgresOSM(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> sr_name = 'Rutland'  # name of a subregion
>>> dat_dir = "tests"  # name of a data directory where the subregion data is

>>> # -- Example 1: Import data of a PBF file ------------------------------------

>>> # First, read the PBF data of Rutland
>>> # (If the data file is not available, it'll be downloaded by confirmation)
```

(continues on next page)

(continued from previous page)

```
>>> rutland_pbf_raw = osmdb_test.Reader.read_osm_pbf(sr_name, dat_dir,␣
 ↪verbose=True)
To download .pbf data of the following geographic region(s):
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "tests\" ... Done.

>>> # A quick view of the PBF data
>>> type(rutland_pbf_raw)
dict
>>> list(rutland_pbf_raw.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> # Data of 'points' layer
>>> rutland_pbf_raw_points = rutland_pbf_raw['points']
>>> rutland_pbf_raw_points.head()
                                              points
0  {"type": "Feature", "geometry": {"type": "Poin...
1  {"type": "Feature", "geometry": {"type": "Poin...
2  {"type": "Feature", "geometry": {"type": "Poin...
3  {"type": "Feature", "geometry": {"type": "Poin...
4  {"type": "Feature", "geometry": {"type": "Poin...

>>> # Import all layers of the raw PBF data of Rutland
>>> osmdb_test.import_osm_data(rutland_pbf_raw, table_name=sr_name, verbose=True)
To import data into table "Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
    "points" ... Done: <total of rows> features.
    "lines" ... Done: <total of rows> features.
    "multilinestrings" ... Done: <total of rows> features.
    "multipolygons" ... Done: <total of rows> features.
    "other_relations" ... Done: <total of rows> features.

>>> # Get further-parsed PBF data
>>> rutland_pbf = osmdb_test.Reader.read_osm_pbf(sr_name, dat_dir,
...                                              parse_raw_feat=True,
...                                              transform_geom=True,
...                                              transform_other_tags=True)

>>> type(rutland_pbf)
dict
>>> list(rutland_pbf.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> # Import data of selected layers into specific schemas

>>> schemas = {"schema_0": 'lines',
...            "schema_1": 'points',
...            "schema_2": 'multipolygons'}

>>> osmdb_test.import_osm_data(rutland_pbf, sr_name, schemas, verbose=True)
To import data into table "Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
    "schema_0" ... Done: <total of rows> features.
    "schema_1" ... Done: <total of rows> features.
    "schema_2" ... Done: <total of rows> features.

>>> # To drop the schemas "schema_0", "schema_1" and "schema_2"
```

(continues on next page)

(continued from previous page)

```
>>> osmdb_test.drop_schema(schemas.keys(), verbose=True)
To drop the following schemas from postgres:***@localhost:5432/osmdb_test:
    "schema_0"
    "schema_1"
    "schema_2"
? [No]|Yes: yes
Dropping ...
    "schema_0" ... Done.
    "schema_1" ... Done.
    "schema_2" ... Done.

>>> # Delete the downloaded PBF data file
>>> os.remove(cd(dat_dir, "rutland-latest.osm.pbf"))

>>> # -- Example 2: Import data of a shapefile ----------------------------------

>>> # Read shapefile data of Rutland
>>> rutland_shp = osmdb_test.Reader.read_shp_zip(sr_name, data_dir=dat_dir,
...                                              rm_extracts=True,
...                                              rm_shp_zip=True,
...                                              verbose=True)
To download .shp.zip data of the following geographic region(s):
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest-free.shp.zip" to "tests\" ... Done.
Extracting "tests\rutland-latest-free.shp.zip" ...
to "tests\rutland-latest-free-shp\"
Done.
Deleting the extracts "tests\rutland-latest-free-shp\"  ... Done.
Deleting "tests\rutland-latest-free.shp.zip" ... Done.

>>> # A quick view of the shapefile data
>>> type(rutland_shp)
dict
>>> list(rutland_shp.keys())
['transport',
 'railways',
 'roads',
 'places',
 'landuse',
 'pofw',
 'natural',
 'traffic',
 'pois',
 'water',
 'buildings',
 'waterways']

>>> # Import all layers of the shapefile data of Rutland
>>> osmdb_test.import_osm_data(rutland_shp, table_name=sr_name, verbose=True)
To import data into table "Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
    "transport" ... Done: <total of rows> features.
    "railways" ... Done: <total of rows> features.
    "roads" ... Done: <total of rows> features.
    "places" ... Done: <total of rows> features.
    "landuse" ... Done: <total of rows> features.
    "pofw" ... Done: <total of rows> features.
    "natural" ... Done: <total of rows> features.
```

(continues on next page)

(continued from previous page)

```
    "traffic" ... Done: <total of rows> features.
    "pois" ... Done: <total of rows> features.
    "water" ... Done: <total of rows> features.
    "buildings" ... Done: <total of rows> features.
    "waterways" ... Done: <total of rows> features.

>>> # -- Example 3: Import BBBike shapefile data file of Leeds -------------------

>>> osmdb_test.DataSource = 'BBBike'
>>> sr_name = 'Leeds'

>>> leeds_shp = osmdb_test.Reader.read_shp_zip(sr_name, data_dir=dat_dir,
...                                            rm_extracts=True, rm_shp_zip=True,
...                                            verbose=True)
To download .shp.zip data of the following geographic region(s):
    Leeds
? [No]|Yes: yes
Downloading "Leeds.osm.shp.zip" to "tests\" ... Done.
Extracting "tests\Leeds.osm.shp.zip" ...
to "tests\"
Done.
Parsing files at "tests\Leeds-shp\shape\" ... Done.
Deleting the extracts "tests\Leeds-shp\" ... Done.
Deleting "tests\Leeds.osm.shp.zip" ... Done.

>>> # A quick view of the shapefile data
>>> type(leeds_shp)
dict
>>> list(leeds_shp.keys())
['points',
 'railways',
 'roads',
 'places',
 'landuse',
 'natural',
 'buildings',
 'waterways']

>>> # Import all layers of the shapefile data of Leeds
>>> osmdb_test.import_osm_data(leeds_shp, table_name=sr_name, verbose=True)
To import data into table "Leeds" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing the data ...
    "points" ... Done: <total of rows> features.
    "railways" ... Done: <total of rows> features.
    "roads" ... Done: <total of rows> features.
    "places" ... Done: <total of rows> features.
    "landuse" ... Done: <total of rows> features.
    "natural" ... Done: <total of rows> features.
    "buildings" ... Done: <total of rows> features.
    "waterways" ... Done: <total of rows> features.

>>> # Delete the database 'osmdb_test'
>>> osmdb_test.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.import_osm_layer

PostgresOSM.**import_osm_layer**(*osm_layer_data*, *table_name*, *schema_name*, *table_named_as_subregion=False*, *schema_named_as_layer=False*, *if_exists='replace'*, *force_replace=False*, *chunk_size=None*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Import one layer of OSM data into a table.

Parameters

- **osm_layer_data** (*pandas.DataFrame or geopandas.GeoDataFrame*) – one layer of OSM data
- **schema_name** (*str*) – name of a schema (or name of a PBF layer)
- **table_name** (*str*) – name of a table
- **table_named_as_subregion** (*bool*) – whether to use subregion name as a table name, defaults to `False`
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name, defaults to `False`
- **if_exists** (*str*) – if the table already exists, to `'replace'` (default), `'append'` or `'fail'`
- **force_replace** (*bool*) – whether to force to replace 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`
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `False`
- **kwargs** – optional parameters of pyhelpers.sql.PostgreSQL.dump_data

Examples:

```
>>> import os
>>> from pyhelpers.dir import cd
>>> from pydriosm.ios import PostgresOSM

>>> osmdb_test = PostgresOSM(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> sr_name = 'Rutland'  # name of a subregion
>>> dat_dir = "tests"  # name of a data directory where the subregion data is

>>> # -- Example 1: Import data of the 'points' layer of a PBF file --------------

>>> # First, read the PBF data of Rutland (from Geofabrik free download server)
>>> # (If the data file is not available, it'll be downloaded by confirmation)
```

(continues on next page)

(continued from previous page)

```
>>> rutland_pbf_raw = osmdb_test.Reader.read_osm_pbf(sr_name, dat_dir,␣
 ↪verbose=True)
To download .pbf data of the following geographic region(s):
    London
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "tests\" ... Done.

>>> # A quick view of the PBF data
>>> type(rutland_pbf_raw)
dict
>>> list(rutland_pbf_raw.keys())
['points', 'lines', 'multilinestrings', 'multipolygons', 'other_relations']

>>> # Get the data of 'points' layer
>>> rutland_pbf_raw_points = rutland_pbf_raw['points']
>>> rutland_pbf_raw_points.head()
                                              points
0  {"type": "Feature", "geometry": {"type": "Poin...
1  {"type": "Feature", "geometry": {"type": "Poin...
2  {"type": "Feature", "geometry": {"type": "Poin...
3  {"type": "Feature", "geometry": {"type": "Poin...
4  {"type": "Feature", "geometry": {"type": "Poin...

>>> # Use the region name as a table name for storing the data in PostgreSQL server
>>> tbl = sr_name
>>> # Use a default layer (key) name as a schema name
>>> schema = list(rutland_pbf_raw.keys())[0]  # 'points'

>>> # Now import the data of 'points' into the PostgreSQL server
>>> osmdb_test.import_osm_layer(rutland_pbf_raw_points, tbl, schema, verbose=True)
To import data into "points"."Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Creating a schema: "points" ... Done.
Importing the data into the table "points"."Rutland" ... Done.

>>> tbl_col_info = osmdb_test.get_subregion_table_column_info(tbl, schema)
>>> tbl_col_info.head()
                   column_0
table_catalog     osmdb_test
table_schema          points
table_name           Rutland
column_name           points
ordinal_position           1

>>> rutland_pbf_parsed = osmdb_test.Reader.read_osm_pbf(sr_name, dat_dir,
...                                                     parse_raw_feat=True,
...                                                     transform_geom=True)

>>> # Get the parsed data of 'points' layer
>>> rutland_pbf_points = rutland_pbf_parsed[schema]
>>> rutland_pbf_points.head()
         id  ...                    other_tags
0    488432  ...              "odbl"=>"clean"
1    488658  ...                          None
2  13883868  ...                          None
3  14049101  ...  "traffic_calming"=>"cushion"
4  14558402  ...      "direction"=>"clockwise"
[5 rows x 12 columns]

>>> # Import the parsed 'points' data into the PostgreSQL server
```

(continues on next page)

(continued from previous page)

```
>>> osmdb_test.import_osm_layer(rutland_pbf_points, tbl, schema, verbose=True)
To import data into "points"."Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
The table "points"."Rutland" already exists and is replaced.
Importing the data into the table "points"."Rutland" ... Done.

>>> # Delete the downloaded PBF data file
>>> os.remove(cd(dat_dir, "rutland-latest.osm.pbf"))

>>> # -- Example 2: Import data of the 'railways' layer of a shapefile -----------

>>> # Read the data of 'railways' layer
>>> # (without retaining any downloaded shapefile and extracts)
>>> lyr_name = 'railways'

>>> rutland_railways_shp = osmdb_test.Reader.read_shp_zip(
...     subregion_name=sr_name, layer_names=lyr_name, data_dir=dat_dir,
...     rm_extracts=True, rm_shp_zip=True, verbose=True)
To download .shp.zip data of the following geographic region(s):
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest-free.shp.zip" to "tests\" ... Done.
Extracting the following layer(s):
    'railways'
from "tests\rutland-latest-free.shp.zip" ...
to "tests\rutland-latest-free-shp\"
Done.
Deleting the extracts "tests\rutland-latest-free-shp\"  ... Done.
Deleting "tests\rutland-latest-free.shp.zip" ... Done.

>>> type(rutland_railways_shp)
dict
>>> list(rutland_railways_shp.keys())
# ['railways']

>>> # Get the data of 'railways' layer
>>> rutland_railways_shp_ = rutland_railways_shp[lyr_name]

>>> rutland_railways_shp_.head()
    osm_id  code  ...                                        coordinates shape_type
0  2162114  6101  ...  [(-0.4528083, 52.6993402), (-0.4518933, 52.698...          3
1  3681043  6101  ...  [(-0.6531215, 52.5730787), (-0.6531793, 52.572...          3
2  3693985  6101  ...  [(-0.7323403, 52.6782102), (-0.7319059, 52.678...          3
3  3693986  6101  ...  [(-0.6173072, 52.6132317), (-0.6241869, 52.614...          3
4  4806329  6101  ...  [(-0.4576926, 52.7035194), (-0.4565358, 52.702...          3
[5 rows x 9 columns]

>>> osmdb_test.import_osm_layer(rutland_railways_shp_, table_name=sr_name,
...                             schema_name=lyr_name, verbose=True)
To import data into "railways"."Rutland" at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Creating a schema: "railways" ... Done.
Importing the data into the table "railways"."Rutland" ... Done.

>>> tbl_col_info = osmdb_test.get_subregion_table_column_info(tbl, lyr_name)
>>> tbl_col_info.head()
                   column_0    column_1  ...     column_7    column_8
table_catalog     osmdb_test  osmdb_test  ...   osmdb_test  osmdb_test
table_schema        railways    railways  ...     railways    railways
table_name           Rutland     Rutland  ...      Rutland     Rutland
```

(continues on next page)

(continued from previous page)

```
column_name           osm_id        code  ...  coordinates  shape_type
ordinal_position           1           2  ...            8           9
[5 rows x 9 columns]

>>> # Delete the database 'osmdb_test'
>>> osmdb_test.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.import_subregion_osm_pbf

PostgresOSM.**import_subregion_osm_pbf**(*subregion_names*, *data_dir=None*, *update_osm_pbf=False*, *if_exists='replace'*, *chunk_size_limit=50*, *parse_raw_feat=False*, *transform_geom=False*, *transform_other_tags=False*, *pickle_pbf_file=False*, *rm_osm_pbf=False*, *confirmation_required=True*, *verbose=False*, ***kwargs*)

Import data of geographic region(s) that do not have (sub-)subregions into a database.

Parameters

- **subregion_names** (*str or list or None*) – name(s) of geographic region(s)
- **data_dir** (*str or None*) – directory where the PBF data file is located/saved; if None (default), the default directory
- **update_osm_pbf** (*bool*) – whether to update .osm.pbf data file (if available), defaults to False
- **if_exists** (*str*) – if the table already exists, to 'replace' (default), 'append' or 'fail'
- **chunk_size_limit** (*int*) – threshold (in MB) that triggers the use of chunk parser, defaults to 50; if the size of the .osm.pbf file (in MB) is greater than chunk_size_limit, it will be parsed in a chunk-wise way
- **parse_raw_feat** (*bool*) – whether to parse each feature in the raw data, defaults to False
- **transform_geom** (*bool*) – whether to transform a single coordinate (or a collection of coordinates) into a geometric object, defaults to False
- **transform_other_tags** (*bool*) – whether to transform 'other_tags' into a dictionary, defaults to False
- **pickle_pbf_file** (*bool*) – whether to save the .pbf data as a .pickle file, defaults to False

- **rm_osm_pbf** (*bool*) – whether to delete the downloaded .osm.pbf file, defaults to `False`
- **confirmation_required** (*bool*) – whether to ask 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 `.import_osm_pbf_layer()`

Examples:

```
>>> import os
>>> from pyhelpers.dir import cd
>>> from pyhelpers.store import load_pickle
>>> from pydriosm.ios import PostgresOSM

>>> osmdb_test = PostgresOSM(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> # -- Example 1: Import PBF data of Rutland -----------------------------------

>>> sr_name = 'Rutland'  # name of a subregion
>>> dat_dir = "tests"  # name of a data directory where the subregion data is

>>> osmdb_test.import_subregion_osm_pbf(sr_name, dat_dir, rm_osm_pbf=True,
...                                     verbose=True)
To import .osm.pbf data of the following geographic region(s) into postgres:***@...
 ↪:
    Rutland
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "tests\" ... Done.
mporting the data into table "Rutland" ...
    "points" ... Done: <total of rows> features.
    "lines" ... Done: <total of rows> features.
    "multilinestrings" ... Done: <total of rows> features.
    "multipolygons" ... Done: <total of rows> features.
    "other_relations" ... Done: <total of rows> features.
Deleting "tests\rutland-latest.osm.pbf" ... Done.

>>> # -- Example 2: Import PBF data of Victoria and Waterloo ---------------------

>>> # The PBF data of Victoria and Waterloo is available on BBBike download server
>>> osmdb_test.DataSource = 'BBBike'
>>> sr_names = ['Victoria', 'Waterloo']

>>> # Note this may take a few minutes or even longer
>>> osmdb_test.import_subregion_osm_pbf(
...     sr_names, dat_dir, parse_raw_feat=True, transform_geom=True,
...     transform_other_tags=True, pickle_pbf_file=True, rm_osm_pbf=True,
...     verbose=True)
To import .osm.pbf data of the following geographic region(s) into postgres:***@...
 ↪:
    Victoria
    Waterloo
? [No]|Yes: yes
Downloading "Victoria.osm.pbf" to "tests\" ... Done.
Parsing "tests\Victoria.osm.pbf" ... Done.
Importing the data into table "Victoria" ...
    "points" ... Done: <total of rows> features.
```

(continues on next page)

(continued from previous page)

```
    "lines" ... Done: <total of rows> features.
    "multilinestrings" ... Done: <total of rows> features.
    "multipolygons" ... Done: <total of rows> features.
    "other_relations" ... Done: <total of rows> features.
Saving "Victoria-pbf.pickle" to "tests" ... Done.
Deleting "tests\Victoria.osm.pbf" ... Done.
Downloading "Waterloo.osm.pbf" to "tests\" ... Done.
Parsing "tests\Waterloo.osm.pbf" ... Done.
Importing the data into table "Waterloo" ...
    "points" ... Done: <total of rows> features.
    "lines" ... Done: <total of rows> features.
    "multilinestrings" ... Done: <total of rows> features.
    "multipolygons" ... Done: <total of rows> features.
    "other_relations" ... Done: <total of rows> features.
Saving "Waterloo-pbf.pickle" to "tests\" ... Done.
Deleting "tests\Waterloo.osm.pbf" ... Done.

>>> # Since `pickle_pbf_file` was set to be True,
>>> # the parsed PBF data have been saved as Pickle files

>>> # Data of Victoria
>>> victoria_pbf = load_pickle(cd(dat_dir, "Victoria-pbf.pickle"))

>>> # Data of the 'points' layer of Victoria
>>> victoria_pbf_points = victoria_pbf['points']
>>> victoria_pbf_points.head()
         id                       coordinates  ... man_made other_tags
0  25832817  POINT (-123.3101944 48.4351988)  ...     None       None
1  25832849  POINT (-123.3162637 48.4336654)  ...     None       None
2  25832953  POINT (-123.3157486 48.4309841)  ...     None       None
3  25832954  POINT (-123.3209478 48.4324002)  ...     None       None
4  25832995    POINT (-123.322405 48.432167)  ...     None       None
[5 rows x 12 columns]

>>> # Data of Waterloo
>>> waterloo_pbf = load_pickle(cd(dat_dir, "Waterloo-pbf.pickle"))

>>> # Data of the 'points' layer of Victoria
>>> waterloo_pbf_points = waterloo_pbf['points']
>>> waterloo_pbf_points.head()
         id  ...                                 other_tags
0  10782939  ...                                       None
1  10782965  ...                                       None
2  14509209  ...                                       None
3  14657092  ...  {'traffic_signals:direction': 'backward'}
4  14657140  ...                                       None
[5 rows x 12 columns]

>>> # Delete the Pickle files
>>> os.remove(cd(dat_dir, "Victoria-pbf.pickle"))
>>> os.remove(cd(dat_dir, "Waterloo-pbf.pickle"))

>>> # Delete the database 'osmdb_test'
>>> osmdb_test.drop_database(verbose=True)
To drop the database "osmdb_test" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "osmdb_test" ... Done.
```

PostgresOSM.list_table_names

PostgresOSM.**list_table_names**(*schema_name='public'*, *verbose=False*)

List the names of all tables in a schema.

Parameters

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

Returns a list of table names

Return type list or None

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> lst_tbl_names = testdb.list_table_names()
>>> print(lst_tbl_names)
[]

>>> lst_tbl_names = testdb.list_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_table_name = 'test_table'
>>> column_specs = 'col_name_1 INT, col_name_2 TEXT'
>>> testdb.create_table(new_table_name, column_specs, verbose=True)
Creating a table: "public"."test_table" ... Done.

>>> lst_tbl_names = testdb.list_table_names(schema_name='public')
>>> print(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.
```

PostgresOSM.psql_insert_copy

static PostgresOSM.**psql_insert_copy**(*sql_table*, *sql_db_engine*, *column_name_list*, *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_name_list** – (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 [SQL-P-PIC-1].

PostgresOSM.read_sql_query

PostgresOSM.**read_sql_query**(*sql_query*, *method='tempfile'*, *tempfile_mode='w+b'*, *max_size_spooled=1*, *delimiter=','*, *dtype=None*, *tempfile_kwargs=None*, *stringio_kwargs=None*, ***kwargs*)

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

See also [SQL-P-RSQ-1], [SQL-P-RSQ-2] and [SQL-P-RSQ-3].

Parameters

- **sql_query** (*str*) – 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
- **tempfile_mode** (*str*) – mode of the specified *method*, defaults to `'w+b'`
- **max_size_spooled** (*int or float*) – `max_size` of tempfile.SpooledTemporaryFile, defaults to `1` (in gigabyte)
- **delimiter** (*str*) – delimiter used in data, defaults to `','`
- **dtype** (*dict or None*) – data type for specified data columns, *dtype* used by pandas.read_csv, defaults to `None`
- **tempfile_kwargs** – optional parameters of tempfile.TemporaryFile or tempfile.SpooledTemporaryFile
- **stringio_kwargs** – 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:

```
>>> import pandas
>>> from pyhelpers.sql import PostgreSQL

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

(continues on next page)

(continued from previous page)

```
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> # Create a pandas.DataFrame
>>> xy_array = [(530034, 180381),
...             (406689, 286822),
...             (383819, 398052),
...             (582044, 152953)]
>>> idx_labels = ['London', 'Birmingham', 'Manchester', 'Leeds']
>>> col_names = ['Easting', 'Northing']
>>> dat = pandas.DataFrame(xy_array, index=idx_labels, columns=col_names)

>>> print(dat)
            Easting  Northing
London       530034    180381
Birmingham   406689    286822
Manchester   383819    398052
Leeds        582044    152953

>>> table = 'England'
>>> schema = 'points'

>>> testdb.import_data(dat, table, schema, if_exists='replace', index=True,
...                    chunk_size=None, force_replace=False, col_type=None,␣
 ↪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.

>>> res = testdb.table_exists(table, schema)
>>> print("The table "{}"."{}" exists? {}.".format(schema, table, res))
The table "points"."England" exists? True.

>>> dat_retrieval = testdb.read_table(table, schema, index_col='index')
>>> dat_retrieval.index.name = None
>>> print(dat_retrieval)
            Easting  Northing
London       530034    180381
Birmingham   406689    286822
Manchester   383819    398052
Leeds        582044    152953

>>> # Alternatively
>>> sql_qry = 'SELECT * FROM "{}"."{}"'.format(schema, table)

>>> dat_retrieval_alt = testdb.read_sql_query(sql_qry, index_col='index')
>>> dat_retrieval_alt.index.name = None
>>> print(dat_retrieval_alt)
            Easting  Northing
London       530034    180381
Birmingham   406689    286822
Manchester   383819    398052
Leeds        582044    152953

>>> # Delete the table "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.
```

(continues on next page)

(continued from previous page)

```
>>> # Delete the schema "points"
>>> testdb.drop_schema(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

sql_qry = 'SELECT * FROM "table_name" '
          'WHERE "timestamp_column_name" BETWEEN %(ts_start)s AND %(ts_end)s'

params = {'ds_start': datetime.datetime.today(), 'ds_end': datetime.datetime.
 ↪today()}

data_frame = pandas.read_sql(sql_qry, testdb.engine, params=params)
```

PostgresOSM.read_table

PostgresOSM.**read_table**(*table_name*, *schema_name='public'*, *condition=None*, *chunk_size=None*, *sorted_by=None*, ***kwargs*)

Read data from a table of the database being connected.

See also [SQL-P-RT-1].

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str*) – name of a schema, defaults to `'public'`
- **condition** (*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 pandas.read_sql

Returns data frame from the specified table

Return type pandas.DataFrame

See the example for the method `.read_sql_query()`.

PostgresOSM.schema_exists

PostgresOSM.**schema_exists**(*schema_name*)

Check if a schema exists in the PostgreSQL server being connected.

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

Returns True if the schema exists, False otherwise

Return type bool

Example:

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

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

PostgresOSM.subregion_table_exists

PostgresOSM.**subregion_table_exists**(*subregion_name*, *layer_name*, *table_named_as_subregion=False*, *schema_named_as_layer=False*)

Check if a table (for a geographic region) exists.

Parameters

- **subregion_name** (*str*) – name of a geographic region, which acts as a table name
- **layer_name** (*str*) – name of an OSM layer (e.g. 'points', 'railways', ...), which acts as a schema name
- **table_named_as_subregion** (*bool*) – whether to use subregion name as table name, defaults to False
- **schema_named_as_layer** (*bool*) – whether a schema is named as a layer name, defaults to False

Returns True if the table exists, False otherwise

Return type bool

Examples:

```
>>> from pydriosm.ios import PostgresOSM

>>> osmdb_test = PostgresOSM(database_name='osmdb_test')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.
```

(continues on next page)

(continued from previous page)

```
>>> region_name = 'London'
>>> lyr_name = 'pt'

>>> # (Suppose the table, pt."London" is available.)
>>> osmdb_test.subregion_table_exists(region_name, lyr_name)
True

>>> # (Suppose the table, points."Greater London" is available.)
>>> osmdb_test.subregion_table_exists(region_name, lyr_name,
...                                   table_named_as_subregion=True,
...                                   schema_named_as_layer=True)
True
```

PostgresOSM.table_exists

PostgresOSM.**table_exists**(*table_name*, *schema_name='public'*)

Check if a table exists in the database being connected.

Parameters

- **table_name** (*str*) – name of a table
- **schema_name** (*str*) – name of a schema, defaults to 'public'

Returns True if the table exists, False otherwise

Return type bool

Examples:

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

>>> testdb = PostgreSQL('localhost', 5432, username='postgres', database_name=
↪'testdb')
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/testdb ... Successfully.

>>> testdb.table_exists(table_name='England', schema_name='points')
>>> # (if 'points.England' does not exist)
False

>>> # Delete the database "testdb"
>>> testdb.drop_database(verbose=True)
To drop the database "testdb" from postgres:***@localhost:5432
? [No]|Yes: yes
Dropping "testdb" ... Done.
```

Attributes

Downloader	An instance of either *GeofabrikDownloader* or *BBBikeDownloader*, depending on the specified data_source for creating an instance of *PostgresOSM* instance.

continues on next page

Table 15 – continued from previous page	
Name	Name of the current *PostgresOSM.Downloader*.
Reader	An instance of either *GeofabrikReader* or *BBBikeReader*, depending on the specified `data_source` for creating an instance of *PostgresOSM*.
URL	Homepage URL of data resource for current *PostgresOSM.Downloader*.

PostgresOSM.Downloader

property PostgresOSM.**Downloader**

An instance of either *GeofabrikDownloader* or *BBBikeDownloader*, depending on the specified `data_source` for creating an instance of *PostgresOSM* instance.

Example:

```
>>> from pydriosm.ios import PostgresOSM

>>> database_name = 'osmdb_test'

>>> osmdb_test = PostgresOSM(database_name=database_name)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> type(osmdb_test.Downloader)
pydriosm.downloader.GeofabrikDownloader
```

PostgresOSM.Name

property PostgresOSM.**Name**

Name of the current *PostgresOSM.Downloader*.

Example:

```
>>> from pydriosm.ios import PostgresOSM

>>> database_name = 'osmdb_test'

>>> osmdb_test = PostgresOSM(database_name=database_name)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> print(osmdb_test.Name)
Geofabrik OpenStreetMap data extracts
```

PostgresOSM.Reader

property PostgresOSM.**Reader**

An instance of either *GeofabrikReader* or *BBBikeReader*, depending on the specified `data_source` for creating an instance of *PostgresOSM*.

Example:

```
>>> from pydriosm.ios import PostgresOSM

>>> database_name = 'osmdb_test'

>>> osmdb_test = PostgresOSM(database_name=database_name)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> type(osmdb_test.Reader)
pydriosm.reader.GeofabrikReader
```

PostgresOSM.URL

property PostgresOSM.**URL**

Homepage URL of data resource for current *PostgresOSM.Downloader*.

Example:

```
>>> from pydriosm.ios import PostgresOSM

>>> database_name = 'osmdb_test'

>>> osmdb_test = PostgresOSM(database_name=database_name)
Password (postgres@localhost:5432): ***
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> print(osmdb_test.URL)
https://download.geofabrik.de/
```

Functions

get_default_layer_name(schema_name)	Get default name (as an input schema name) of an OSM layer for the class *PostgresOSM*.
validate_schema_names([schema_names, ...])	Validate schema names for importing data into a PostgreSQL database.
validate_table_name(table_name[, sub_space])	Validate a table name for importing OSM data into a PostgreSQL database.

3.3.2 get_default_layer_name

pydriosm.ios.**get_default_layer_name**(*schema_name*)

Get default name (as an input schema name) of an OSM layer for the class *PostgresOSM*.

See, for example, the method *pydriosm.ios.PostgresOSM.import_osm_layer()*.

Parameters **schema_name** (*str*) – name of a schema (or name of an OSM layer)

Returns default name of the layer

Return type str

Example:

```
>>> from pydriosm.ios import get_default_layer_name

>>> lyr_name = get_default_layer_name(schema_name='point')

>>> print(lyr_name)
points
```

3.3.3 validate_schema_names

pydriosm.ios.**validate_schema_names**(*schema_names=None*, *schema_named_as_layer=False*)

Validate schema names for importing data into a PostgreSQL database.

Parameters

- **schema_names** (*typing.Iterable or None*) – one or multiple names of layers, e.g. 'points', 'lines', defaults to None
- **schema_named_as_layer** (*bool*) – whether to use default PBF layer name as the schema name, defaults to False

Returns valid names of the schemas in the database

Return type list

Examples:

```
>>> from pydriosm.ios import validate_schema_names

>>> valid_names = validate_schema_names()

>>> print(valid_names)
[]

>>> input_schema_names = ['point', 'polygon']

>>> valid_names = validate_schema_names(input_schema_names)
>>> print(valid_names)
['point', 'polygon']

>>> valid_names = validate_schema_names(input_schema_names, schema_named_as_layer=True)
>>> print(valid_names)
['points', 'multipolygons']
```

3.3.4 validate_table_name

pydriosm.ios.**validate_table_name**(*table_name*, *sub_space=''*)

Validate a table name for importing OSM data into a PostgreSQL database.

Parameters

- **table_name** (*str*) – name as input of a table in a PostgreSQL database
- **sub_space** (*str*) – substitute for space

Returns valid name of the table in the database

Return type str

Examples:

```
>>> from pydriosm.ios import validate_table_name

>>> region_name = 'greater london'

>>> valid_table_name = validate_table_name(region_name)
>>> print(valid_table_name)
greater london

>>> region_name = 'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch, Wales'

>>> valid_table_name = validate_table_name(region_name, sub_space='_')
>>> print(valid_table_name)
Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch_W..
```

3.4 utils

Helper functions.

Specify resource homepages

geofabrik_homepage()	Specify the homepage URL of the free Geofabrik data extracts.
bbbike_homepage()	Specify the homepage URL of the free BBBike data extracts.

3.4.1 geofabrik_homepage

pydriosm.utils.**geofabrik_homepage**()

Specify the homepage URL of the free Geofabrik data extracts.

Returns URL of the data source homepage

Return type str

3.4.2 bbbike_homepage

pydriosm.utils.**bbbike_homepage**()

Specify the homepage URL of the free BBBike data extracts.

Returns URL of the data source homepage

Return type str

Specify directory/file paths

cd_dat(*sub_dir[, dat_dir, mkdir])	Change directory to `dat_dir` and its sub-directories within a package.
cd_dat_geofabrik(*sub_dir[, mkdir])	Change directory to `dat_Geofabrik` and its sub-directories within a package.
cd_dat_bbbike(*sub_dir[, mkdir])	Change directory to `dat_BBBike` and its sub-directories.

3.4.3 cd_dat

pydriosm.utils.**cd_dat**(**sub_dir*, *dat_dir='dat'*, *mkdir=False*, ***kwargs*)

Change directory to `dat_dir` and its sub-directories within a package.

Parameters

- **sub_dir** (*str*) – name of directory; names of directories (and/or a filename)
- **dat_dir** (*str*) – name of a directory to store data, defaults to `"dat"`
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns an absolute path to a directory (or a file) under `data_dir`

Return type str

Example:

```
>>> import os
>>> from pydriosm.utils import cd_dat

>>> path_to_dat = cd_dat()

>>> print(os.path.relpath(path_to_dat))
pydriosm\dat
```

3.4.4 cd_dat_geofabrik

pydriosm.utils.**cd_dat_geofabrik**(**sub_dir*, *mkdir=False*, ***kwargs*)

Change directory to `dat_Geofabrik` and its sub-directories within a package.

Parameters

- **sub_dir** (*str or typing.PathLike*) – name of directory; names of directories (and/or a filename)
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns an absolute path to a directory (or a file) under `data_dir`

Return type str or typing.PathLike

3.4.5 cd_dat_bbbike

pydriosm.utils.**cd_dat_bbbike**(**sub_dir*, *mkdir=False*, ***kwargs*)

Change directory to `dat_BBBike` and its sub-directories.

Parameters

- **sub_dir** (*str*) – name of directory; names of directories (and/or a filename)
- **mkdir** (*bool*) – whether to create a directory, defaults to `False`
- **kwargs** – optional parameters of os.makedirs, e.g. `mode=0o777`

Returns an absolute path to a directory (or a file) under `data_dir`

Return type str

Specify geometric object types/names

get_pbf_layer_feat_types_dict()	A dictionary for PBF layers and the corresponding geometry types.
get_shp_shape_types_dict()	A dictionary for shape types of shapefiles.
get_shp_shape_types_geom_dict()	A dictionary for shapefiles' shape types and their corresponding geometry objects' types.
get_valid_shp_layer_names()	Get valid layer names of OSM shapefiles.

3.4.6 get_pbf_layer_feat_types_dict

pydriosm.utils.**get_pbf_layer_feat_types_dict**()

A dictionary for PBF layers and the corresponding geometry types.

Returns a dictionary with keys and values being PBF layers and geometry types

Return type dict

3.4.7 get_shp_shape_types_dict

pydriosm.utils.**get_shp_shape_types_dict**()

A dictionary for shape types of shapefiles.

Returns a dictionary with keys and values being codes and shape types

Return type dict

3.4.8 get_shp_shape_types_geom_dict

pydriosm.utils.**get_shp_shape_types_geom_dict**()

A dictionary for shapefiles' shape types and their corresponding geometry objects' types.

Returns a dictionary with keys and values being shape type codes and shapely.geometry types

Return type dict

3.4.9 get_valid_shp_layer_names

pydriosm.utils.**get_valid_shp_layer_names**()

Get valid layer names of OSM shapefiles.

Returns a list of valid layer names of OSM shapefiles

Return type list

Miscellaneous

validate_shp_layer_names(layer_names)	Validate the input of layer name(s) for reading shape files.
find_shp_layer_name(shp_filename)	Find the layer name of OSM shapefile given its filename.
append_fclass_to_filename(shp_filename, ...)	Append a 'fclass' name to the original filename of shapefile.
remove_subregion_osm_file(path_to_osm_f	Remove a downloaded OSM data file.
get_number_of_chunks(path_to_file[, ...])	Compute number of chunks for parsing OSM (mainly PBF) data file in a chunk-wise manner.
convert_dtype_dict()	Specify data-type dictionary for data types of PostgreSQL and pandas.read_csv().

3.4.10 validate_shp_layer_names

pydriosm.utils.**validate_shp_layer_names**(*layer_names*)

Validate the input of layer name(s) for reading shape files.

Parameters **layer_names** (*str or list or None*) – name of a shapefile layer, e.g. 'railways', or names of multiple layers; if `None` (default), returns an empty list; if `'all'`, returns a list of all available layers

Returns valid layer names to be input

Return type list

Examples:

```
>>> from pydriosm.utils import validate_shp_layer_names

>>> lyr_names = None
>>> lyr_names_ = validate_shp_layer_names(lyr_names)
>>> print(lyr_names_)
[]

>>> lyr_names = 'point'
>>> lyr_names_ = validate_shp_layer_names(lyr_names)
>>> print(lyr_names_)
['points']

>>> lyr_names = ['point', 'land']
>>> lyr_names_ = validate_shp_layer_names(lyr_names)
>>> print(lyr_names_)
['points', 'landuse']

>>> lyr_names = 'all'
>>> lyr_names_ = validate_shp_layer_names(lyr_names)
>>> print(lyr_names_)
['buildings',
 'landuse',
 'natural',
 'places',
 'points',
 'pofw',
 'pois',
 'railways',
 'roads',
 'traffic',
 'transport',
 'water',
 'waterways']
```

3.4.11 find_shp_layer_name

pydriosm.utils.**find_shp_layer_name**(*shp_filename*)

Find the layer name of OSM shapefile given its filename.

Parameters **shp_filename** (*str*) – filename of a shapefile (.shp)

Returns layer name of the .shp file

Return type str

3.4.12 append_fclass_to_filename

pydriosm.utils.**append_fclass_to_filename**(*shp_filename*, *feature_names*)

Append a `'fclass'` name to the original filename of shapefile.

Parameters

- **shp_filename** (*str*) – original .shp filename
- **feature_names** (*str or list*) – name (or names) of a `fclass` (or multiple `fclass`) in .shp data

Returns updated filename used for saving only the `fclass` data of the original .shp data file

Return type str

3.4.13 remove_subregion_osm_file

pydriosm.utils.**remove_subregion_osm_file**(*path_to_osm_file*, *verbose=True*)

Remove a downloaded OSM data file.

Parameters

- **path_to_osm_file** (*str*) – absolute path to a downloaded OSM data file
- **verbose** (*bool*) – defaults to `True`

3.4.14 get_number_of_chunks

pydriosm.utils.**get_number_of_chunks**(*path_to_file*, *chunk_size_limit=50*)

Compute number of chunks for parsing OSM (mainly PBF) data file in a chunk-wise manner.

Parameters

- **path_to_file** (*str*) – absolute path to a file
- **chunk_size_limit** (*int*) – threshold (in MB) above which the data file is split into chunks, defaults to `50`;

Returns number of chunks

Return type int or None

3.4.15 convert_dtype_dict

pydriosm.utils.**convert_dtype_dict**()

Specify data-type dictionary for data types of PostgreSQL and pandas.read_csv().

Returns a dictionary as data-type convertor between PostgreSQL and pandas.read_csv()

Return type dict

3.5 settings

Default settings for working environment.

gdal_configurations([reset, max_tmpfile_size])	Set GDAL configurations.

3.5.1 gdal_configurations

pydriosm.settings.**gdal_configurations**(*reset=False*, *max_tmpfile_size=5000*)

Set GDAL configurations. See also [GC-1].

Parameters

- **reset** (*bool*) – reset to default settings, defaults to `False`
- **max_tmpfile_size** (*int*) – maximum size of the temporary file, defaults to `5000`

Example:

```
>>> from pydriosm.settings import gdal_configurations

>>> gdal_configurations(max_tmpfile_size=500)
```

3.6 updater

Updating package data.

update_package_data([confirmation_require ...])	Update package data.

3.6.1 update_package_data

pydriosm.updater.**update_package_data**(*confirmation_required=True*, *interval_sec=2*, *verbose=True*)

Update package data.

Parameters

- **confirmation_required** (*bool*) – whether asking for confirmation to proceed, defaults to `True`
- **interval_sec** (*int*) – time gap (in seconds) between the updating of different classes, defaults to `5`
- **verbose** (*bool , int*) – whether to print relevant information in console, defaults to `True`

Example:

```
>>> from pydriosm.updater import update_package_data

>>> update_package_data(confirmation_required=True, verbose=True)
```

(**THE END OF** *Modules*.)

CHAPTER

FOUR

LICENSE

- **PyDriosm**
  - PyDriosm is licensed under GNU General Public License v3.0 (GPLv3).
- **OpenStreetMap data**
  - The free OpenStreetMap data, which is used for the development of PyDriosm, is licensed under the Open Data Commons Open Database License (ODbL) by the OpenStreetMap Foundation (OSMF).
  - For more details about the use of the OpenStreetMap data, refer to the web page of Copyright and Licence.

CHAPTER

FIVE

ACKNOWLEDGEMENT

The development of PyDriosm, together with all the example code for demonstrating how to use the package, is mainly built on free OpenStreetMap data. The author of the package would like to thank all OpenStreetMap contributors who make the data available for free.

PYTHON MODULE INDEX

p

INDEX

I

L

M

N

P

R

S

T

U

V

W