# PyDriosm Documentation

*Release 2.0.2*

**Qian Fu**

**Mar 23, 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: 2.0.2
```

---

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

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

>>> geofabrik_downloader = GeofabrikDownloader()
```

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

```
>>> geofabrik_download_catalogue = geofabrik_downloader.get_download_catalogue()

>>> # Column names
>>> print(geofabrik_download_catalogue.columns.tolist())
['Subregion', 'SubregionURL', '.osm.pbf', '.osm.pbf.Size', '.shp.zip', '.osm.bz2']

>>> print(geofabrik_download_catalogue.head())
      Subregion  ...                                          .osm.bz2
0       Algeria  ...  http://download.geofabrik.de/africa/algeria-la...
```

(continues on next page)

(continued from previous page)

```
1        Angola  ...  http://download.geofabrik.de/africa/angola-lat...
2         Benin  ...  http://download.geofabrik.de/africa/benin-late...
3      Botswana  ...  http://download.geofabrik.de/africa/botswana-l...
4  Burkina Faso  ...  http://download.geofabrik.de/africa/burkina-fa...
[5 rows x 6 columns]
```

If we'd like to download say the protocolbuffer binary format (PBF) data of a specific geographic region, we need to specify the name of the region and the 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"

>>> # Download the OSM PBF data of London from Geofabrik
>>> geofabrik_downloader.download_osm_data(subregion_name, osm_file_format,
...                                        download_dir, verbose=True)
Confirmed 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, we'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, we 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 `"\dat_Geofabrik\Europe\Great Britain\England\"`.

Now we 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, we 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

>>> london_pbf_filename = os.path.basename(path_to_london_pbf)

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

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

Alternatively, we 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'

>>> from pyhelpers.dir import cd

>>> path_to_london_pbf = cd(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, we 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 we 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)
Confirmed 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(path_to_pbf)
<class '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, we could use the class *GeofabrikReader* (see also *pydriosm.reader*).

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

>>> geofabrik_reader = GeofabrikReader()
```

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

To read the PBF data, we 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)
<class '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']

>>> print(rutland_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)
<class 'str'>

>>> rutland_pbf_points_0_ = json.loads(rutland_pbf_points_0)
>>> type(rutland_pbf_points_0_)
<class 'dict'>

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

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

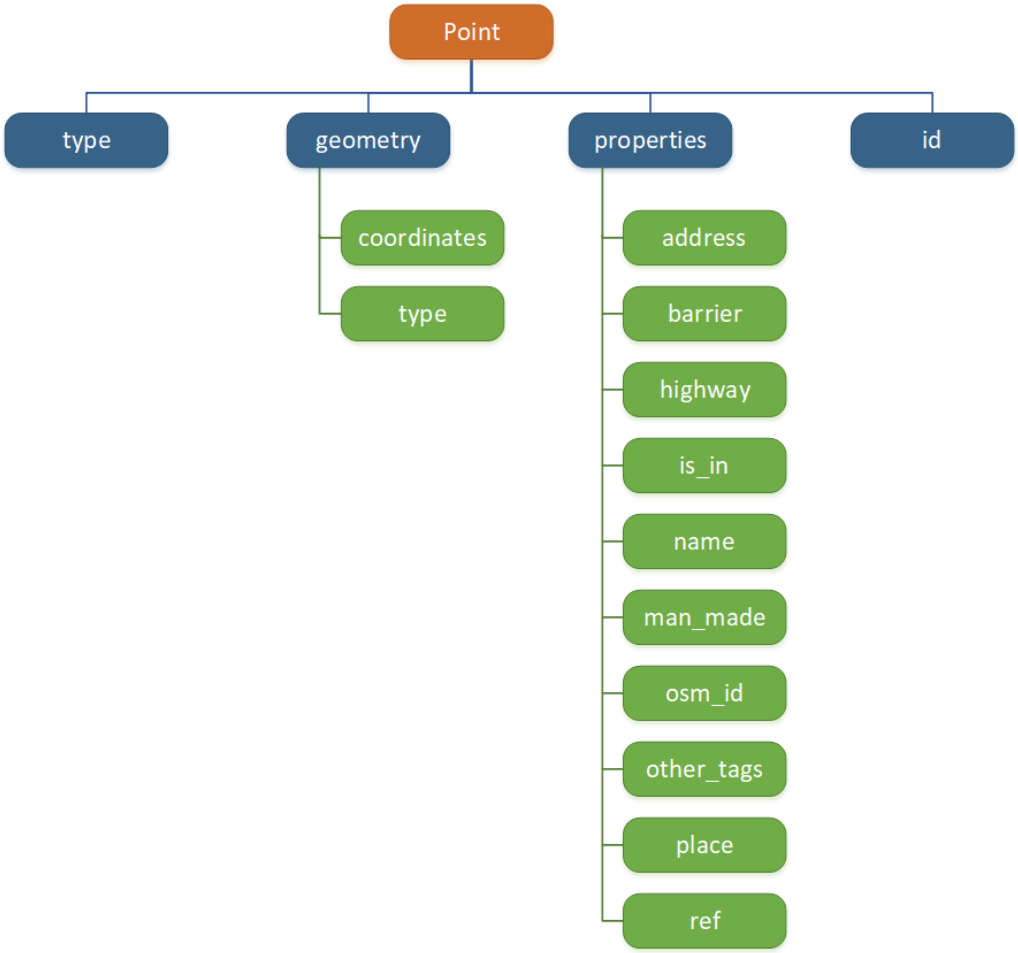


Fig. 1: Type of the geometry object and keys within the nested dictionary of `'points'`

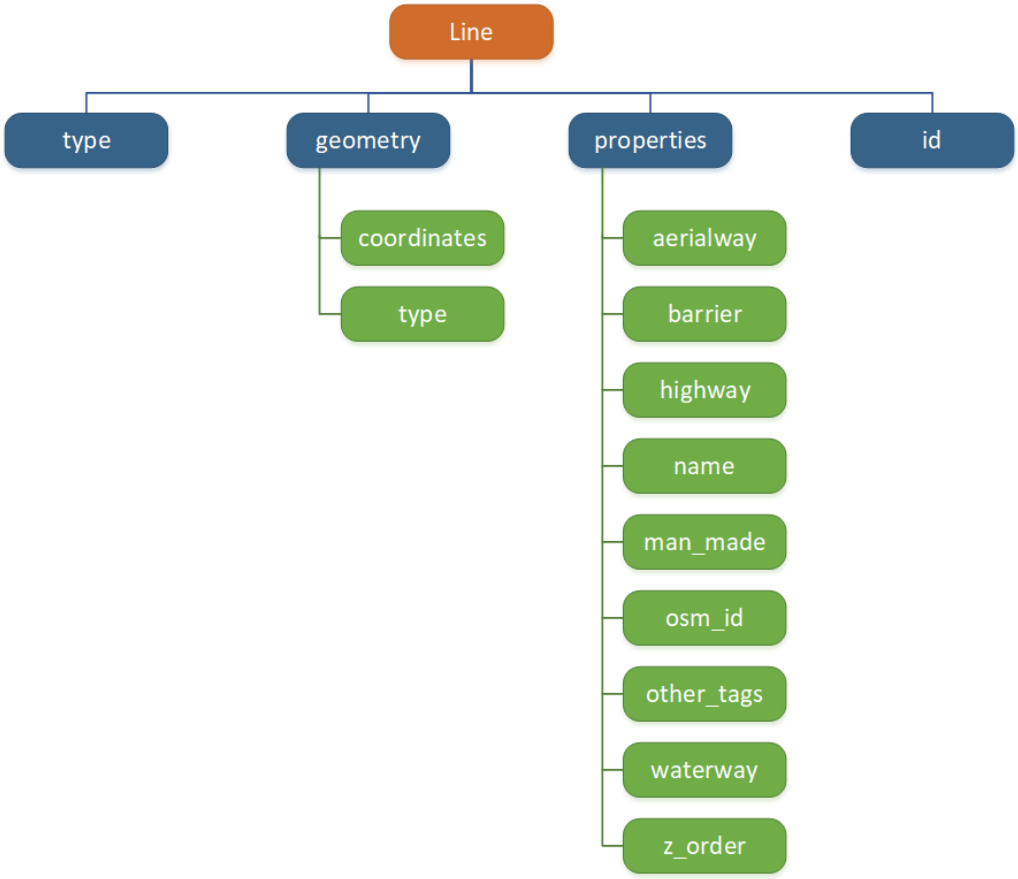


Fig. 2: Type of the geometry object and keys within the nested dictionary of `'lines'`

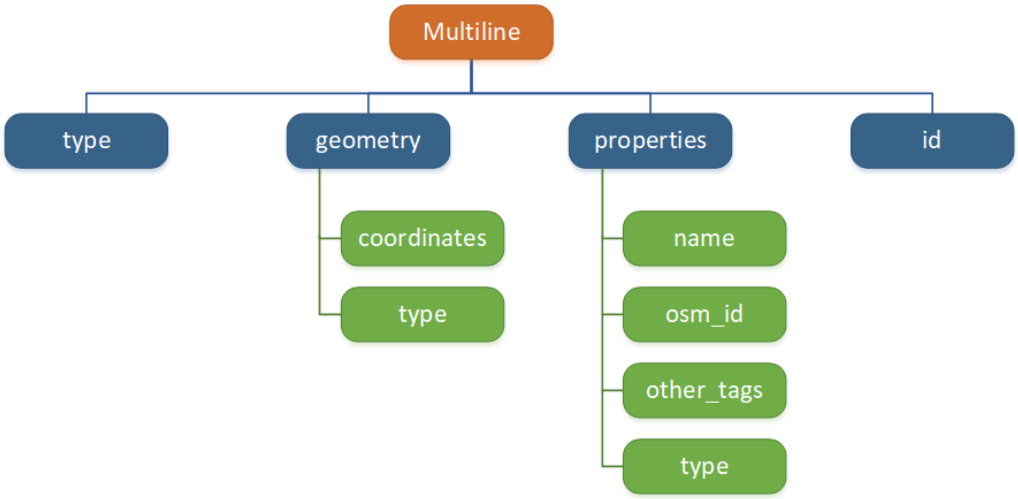


Fig. 3: Type of the geometry object and keys within the nested dictionary of `'multilinestrings'`

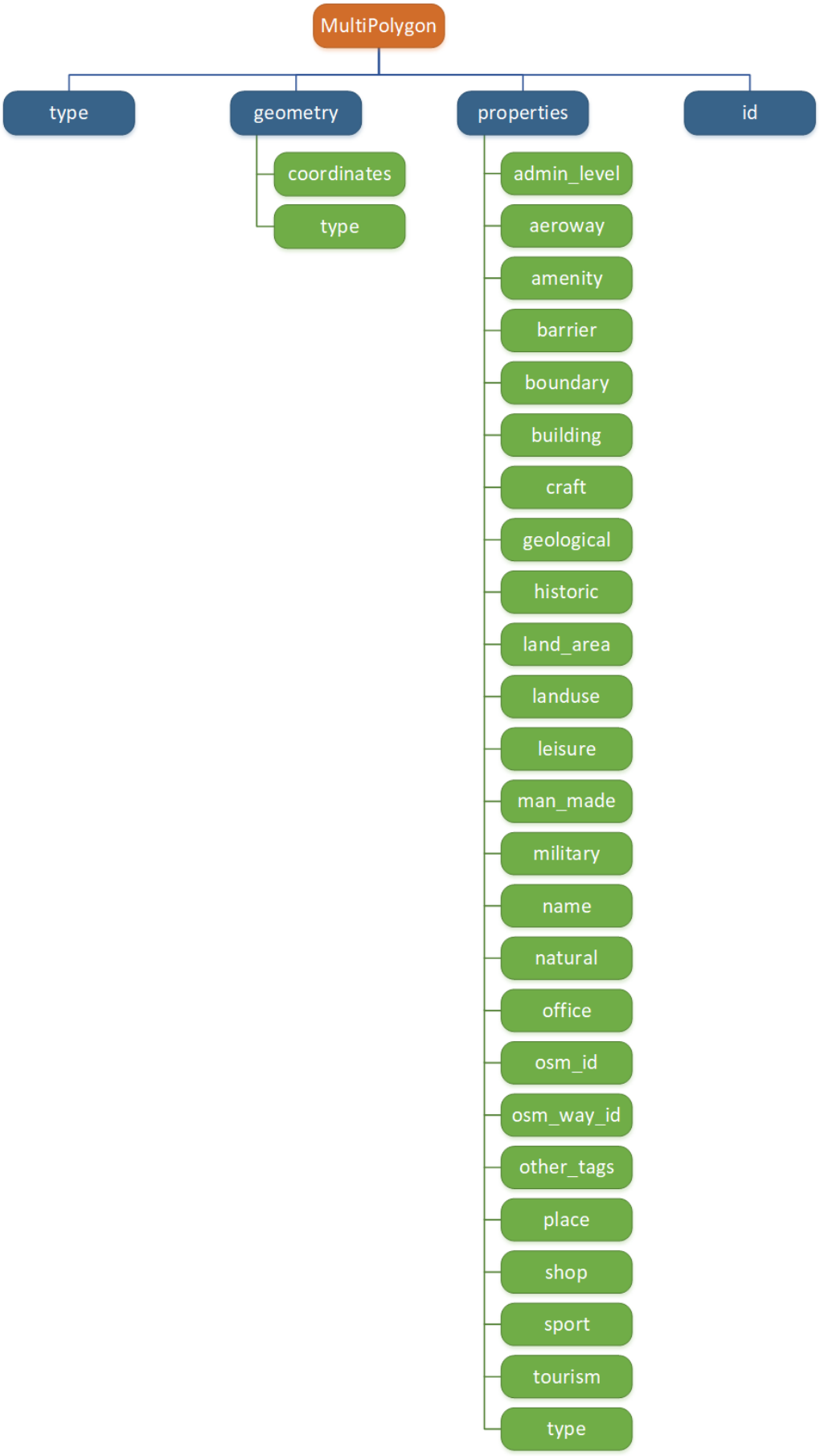


Fig. 4: Type of the geometry object and keys within the nested dictionary of `'multipolygons'`

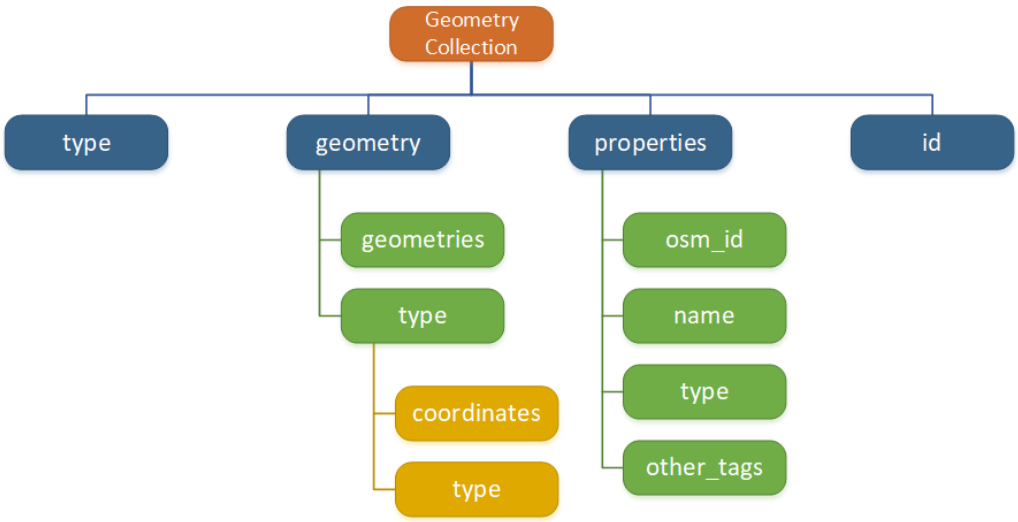


Fig. 5: Type of the geometry object and keys within the nested dictionary of `'other_relations'`

If we set `parse_raw_feat` (which defaults to `False`) to be `True` when reading the PBF data, we 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.

>>> rutland_pbf_parsed_points = rutland_pbf_parsed['points']

>>> print(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  [-0.7249922, 52.6748223]  ...     None  "traffic_calming"=>"cushion"
4  14558402  [-0.7266686, 52.6695051]  ...     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, we 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 we 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, we can use the method *`.read_shp_zip()`*, which depends largely on pyshp or GeoPandas.

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)
Confirmed 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 from "greater-london-latest-free.shp.zip" the following layer(s):
    'railways'
to "\tests\greater-london-latest-free-shp" ...
In progress ... Done.
```

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

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

>>> print(london_railways_shp.head())
   osm_id  code  ... tunnel                                           geometry
0   30804  6101  ...      F    LINESTRING (0.00486 51.62793, 0.00620 51.62927)
1  101298  6103  ...      F  LINESTRING (-0.22496 51.49354, -0.22507 51.494...
2  101486  6103  ...      F  LINESTRING (-0.20555 51.51954, -0.20514 51.519...
3  101511  6101  ...      F  LINESTRING (-0.21189 51.52419, -0.21079 51.523...
4  282898  6103  ...      F  LINESTRING (-0.18626 51.61591, -0.18687 51.61384)
[5 rows x 8 columns]
```

**Note:**

- The parameter `feature_names` is related to `'fclass'` in `london_railways_shp`. We can specify one feature name (or multiple feature names) to get a subset of `london_railways_shp`.
- Similar to *`.read_osm_pbf()`*, if the method *`.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 we'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, we 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(
...     layer_name, subregion_names, data_dir, verbose=True, ret_merged_shp_path=True)
Confirmed to download .shp.zip data of the following geographic region(s):
    Greater London
    Kent
? [No]|Yes: yes
"greater-london-latest-free.shp.zip" of Greater London is already available at "tests".
Downloading "kent-latest-free.shp.zip" to "\tests" ...
Done.
Extracting from "greater-london-latest-free.shp.zip" the following layer(s):
    'railways'
to "\tests\greater-london-latest-free-shp" ...
In progress ... Done.
Extracting from "kent-latest-free.shp.zip" the following layer(s):
    'railways'
to "\tests\kent-latest-free-shp" ...
In progress ... 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 .shp file(s) at "\tests\greater-london_kent_railways".

>>> 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()* (see also *pydriosm.reader*).

## 2.3 Import and fetch data with a PostgreSQL server

Beyond 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, we need to specify the username, password, host address of a PostgreSQL server and name of a database. For example:

```
>>> from pydriosm import PostgresOSM

>>> host = 'localhost'
>>> port = 5432
>>> username = 'postgres'
>>> password = None  # We 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.
```

**Note:**

- If we don't specify a password (for creating the instance `osmdb_test`) as the parameter `password` is `None` by default, we'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 `pydriosm.reader` as properties. In the case of the above instance, `osmdb_test.Downloader` is equivalent to *GeofabrikDownloader*, as the parameter `data_source` is `'Geofabrik'` by default.
- To relate the instance `osmdb_test` to 'BBBike' data, we could 1) recreate an instance by setting `data_source='BBBike'`; or 2) set `osmdb_test.DataSource='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, we can use the method *.import_osm_data()* or *.import_subregion_osm_pbf()*.

For example, let's now try to import `rutland_pbf_parsed` that we 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)
Confirmed to import the data into table "Rutland"
    at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing data into "Rutland" ...
    points ... done: 4253 features.
    lines ... done: 7599 features.
    multilinestrings ... done: 53 features.
    multipolygons ... done: 6382 features.
    other_relations ... done: 13 features.
```

**Note:** The parameter `schema_names` is `None` by default, meaning that we are going to import all of 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. 6); and the data of each layer is imported into a table named as 'Rutland' under the corresponding schema (as illustrated in Fig. 7).

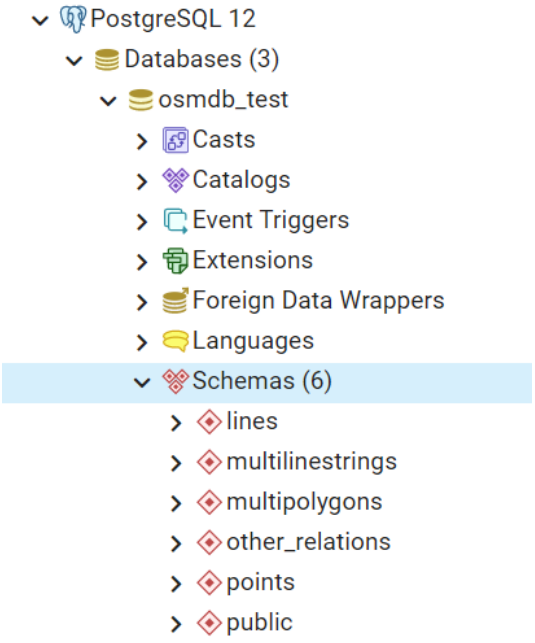


Fig. 6: An illustration of schemas for importing OSM PBF data into a PostgreSQL database

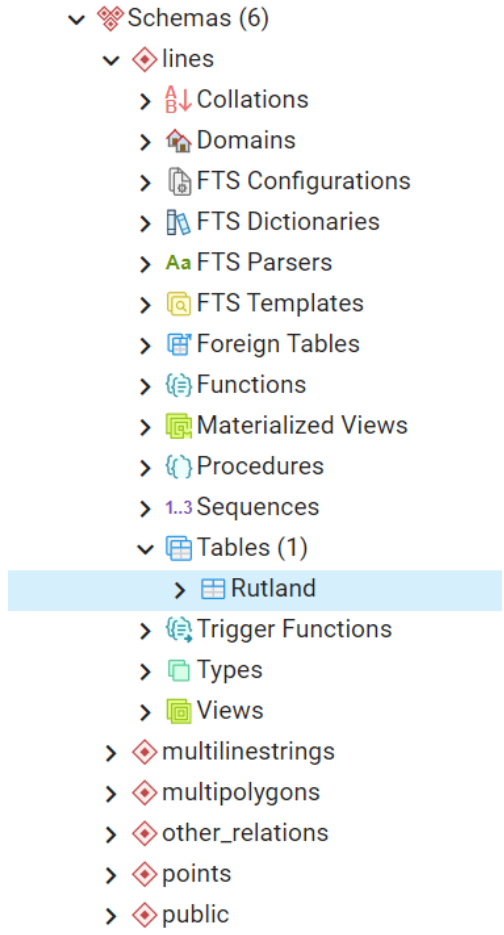


Fig. 7: An illustration of table name for storing the 'lines' layer of the OSM PBF data of Rutland

### 2.3.2 Fetch data from the database

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

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

We 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 we're going to fetch data of all layers available from the database.
- The data stored in the database was parsed by the *geofabrik_reader.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 `.read_osm_pbf()`, we set the parameter `decode_wkt`, which defaults to `False`, to be `True`, so as to retrieve the same data.

---

### 2.3.3 Import/fetch data of specific layers

Of course, we 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. We 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)
Confirmed to download .shp.zip data of the following geographic region(s):
    Birmingham
? [No]|Yes: yes
Downloading "Birmingham.osm.shp.zip" to "\tests" ...
Done.
Extracting all of "Birmingham.osm.shp.zip" to "\tests" ...
```

(continues on next page)

(continued from previous page)

```
In progress ... Done.
Parsing "\tests\Birmingham-shp\shape" ... Done.

# Check names of layers included in the data
>>> print(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, table_name=subregion_name,
...                            schema_names=lyr_names, verbose=True)
Confirmed to import the data into table "Birmingham"
    at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing data into "Birmingham" ...
    railways ... done: 3176 features.
    roads ... done: 119344 features.
    waterways ... done: 2917 features.
```

To fetch only the 'railways' data of Birmingham:

```
>>> lyr_name = 'railways'

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

>>> birmingham_shp_railways_ = birmingham_shp_[lyr_name]

>>> print(birmingham_shp_railways_.head())
    osm_id  ...                                           geometry
0      740  ...  LINESTRING (-1.8178905 52.5700974, -1.8179287 ...
1     2148  ...  LINESTRING (-1.8731878 52.5055513, -1.8727074 ...
2  2950000  ...  LINESTRING (-1.8794134 52.4813762, -1.8795969 ...
3  3491845  ...  LINESTRING (-1.7406017 52.5185831, -1.7394216 ...
4  3981454  ...  LINESTRING (-1.7747469 52.5228419, -1.7744914 ...
[5 rows x 4 columns]
```

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

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

>>> print(birmingham_shp_railways.head())
    osm_id  ...                                           geometry
0      740  ...  LINESTRING (-1.81789 52.57010, -1.81793 52.569...
1     2148  ...  LINESTRING (-1.87319 52.50555, -1.87271 52.505...
2  2950000  ...  LINESTRING (-1.87941 52.48138, -1.87960 52.481...
```

(continues on next page)

(continued from previous page)

```
3  3491845  ...  LINESTRING (-1.74060 52.51858, -1.73942 52.518...
4  3981454  ...  LINESTRING (-1.77475 52.52284, -1.77449 52.522...
[5 rows x 4 columns]
```

**Note:** `birmingham_shp_railways` is a geopandas.GeoDataFrame and `birmingham_shp_railways_` is a pandas.DataFrame. We may have to transform the format of either one to the other before making a comparison between them.

```
>>> import geopandas as gpd

>>> check_equivalence =
...     gpd.GeoDataFrame(birmingham_shp_railways_).equals(birmingham_shp_railways)

>>> print(f"`birmingham_shp_railways_` equals `birmingham_shp_railways`: "
...       f"{check_equivalence}")
`birmingham_shp_railways_` equals `birmingham_shp_railways`: True
```

### 2.3.4 Drop data

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

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

```
>>> osmdb_test.drop_subregion_table(subregion_name, lyr_name, verbose=True)
Confirmed to drop the following table:
    "Birmingham"
  from the following schema:
    "railways"
  at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Dropping 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)
Confirmed to drop the following tables:
    "Birmingham" and
    "Rutland"
  from the following schemas:
    "lines",
    "multilinestrings" and
    "waterways"
  at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Dropping tables ...
    "lines"."Rutland" ... Done.
    "multilinestrings"."Rutland" ... Done.
    "waterways"."Birmingham" ... Done.
```

We could also easily drop the whole database 'osmdb_test' if we don't need it any more:

```
>>> osmdb_test.PostgreSQL.drop_database(verbose=True)
Confirmed 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 before we move on

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:
...     os.remove(cd(data_dir, dat_file))

>>> # # 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 6 modules:

| | |
|---|---|
| *downloader* | Downloading Geofabrik and BBBike OpenStreetMap (OSM) data extracts. |
| *reader* | Reading 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

Downloading Geofabrik and BBBike OpenStreetMap (OSM) data extracts.

### Classes

| | |
|---|---|
| *GeofabrikDownloader*() | A class for downloading OSM data from Geofabrik's free download server. |
| *BBBikeDownloader*() | A class for downloading OSM data from BBBike's free download server. |

### 3.1.1 GeofabrikDownloader

**class** pydriosm.downloader.**GeofabrikDownloader**

A class for downloading OSM data from Geofabrik's free download server.

**Example**:

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

>>> geofabrik_downloader = GeofabrikDownloader()
```

(continues on next page)

(continued from previous page)

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

### 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_na ...) | Download OSM data (in a specific file format) of one (or multiple) geographic region(s) and all its (or their) subregions. |
| *get_continents_subregion_tables*([up ...]) | Get download information for continents. |
| *get_default_osm_filename*(subregion_r ...) | get a default filename for a geograpic region. |
| *get_default_path_to_osm_file*(subreg ...) | 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. |
| *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_nam ...[, ...]) | Make a default directory for downloading data of a geographic region's subregions. |
| *osm_file_exists*(subregion_name, osm_file_format) | Check if a requested data file of a geographic region already exists locally. |
| *search_for_subregions*(*subregion_nan deep]) | Retrieve names of all subregions (if any) of the given geographic region(s). |
| *validate_input_file_format*(osm_file_ | Validate an input file format of OSM data. |
| *validate_input_subregion_name*(subre | 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_sec=None*, *verbose=False*, *ret_download_path=False*)

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

**Parameters**

- **subregion_names** (*str or list*) – name(s) of one (or multiple) geographic region(s) available on Geofabrik's free download server
- **osm_file_format** (*str*) – OSM file format; valid values include ".osm.pbf", ".shp.zip" and ".osm.bz2"
- **download_dir** (*str or None*) – directory for saving the downloaded file(s); if None (default), use the default directory
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **deep_retry** (*bool*) – whether to further check availability of sub-subregions data, defaults to False
- **interval_sec** (*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

**Returns** absolute path(s) to downloaded file(s) when ret_download_path=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
>>> sr_names = ['London', 'Rutland']
>>> file_fmt = ".pbf"

>>> dwnld_paths = geofabrik_downloader.download_osm_data(
```

(continues on next page)

(continued from previous page)

```
...     sr_names, file_fmt, verbose=True, ret_download_path=True)
Confirmed to download .osm.pbf data of the following geographic region(s):
    Greater London
    Rutland
? [No]|Yes: yes
Downloading "greater-london-latest.osm.pbf" to "\dat_... \England" ...
Done.
Downloading "rutland-latest.osm.pbf" to "\dat_... \England" ...
Done.

>>> for dwnld_path in dwnld_paths:
...     print(os.path.relpath(dwnld_path))
dat_GeoFabrik\Europe\Great Britain\England\greater-london-latest.osm.pbf
dat_GeoFabrik\Europe\Great Britain\England\rutland-latest.osm.pbf

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

>>> # Download shapefiles of West Midlands
>>> sr_name = 'west midlands'
>>> file_fmt = ".shp"
>>> dwnld_dir = "tests"

>>> dwnld_path = geofabrik_downloader.download_osm_data(
...     sr_name, file_fmt, dwnld_dir, verbose=True, ret_download_path=True)
Confirmed 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)

>>> # Download shapefiles of Great Britain
>>> sr_name = 'Great Britain'
>>> file_fmt = ".shp"

>>> dwnld_path = geofabrik_downloader.download_osm_data(
...     sr_name, file_fmt, dwnld_dir, deep_retry=True, verbose=True,
...     ret_download_path=True)
Confirmed to download .shp.zip data of the following geographic region(s):
    Great Britain
? [No]|Yes: yes
The .shp.zip data is not found for "Great Britain".
Try downloading the data of its subregions instead [No]|Yes: no

>>> print(dwnld_path)
[]
```

### GeofabrikDownloader.download_subregion_data

GeofabrikDownloader.**download_subregion_data**(*subregion_names*, *osm_file_format*, *download_dir=None*, *update=False*, *verbose=False*, *ret_download_path=False*)

Download OSM data (in a specific file format) of one (or multiple) geographic region(s) and all its (or their) subregions.

**Parameters**

- **subregion_names** (*str or list*) – name(s) of one (or multiple) region(s)/subregion(s) available on Geofabrik's free download server
- **osm_file_format** (*str*) – OSM file format; valid values include ".osm.pbf", ".shp.zip" and ".osm.bz2"
- **download_dir** (*str or None*) – directory for saving the downloaded file(s); if None (default), use the default directory
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **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=True

**Return type** list or str

**Examples**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

>>> file_fmt = ".pbf"
>>> dwnld_dir = "tests"

>>> sr_names = ['rutland', 'west yorkshire']

>>> geofabrik_downloader.download_subregion_data(sr_names, file_fmt,
...                                              dwnld_dir, verbose=True)
Confirmed 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.
```

(continues on next page)

(continued from previous page)

```
>>> os.remove(cd("tests", "rutland-latest.osm.pbf"))

>>> sr_names = ['west midlands', 'west yorkshire']

>>> dwnld_paths = geofabrik_downloader.download_subregion_data(
...     sr_names, file_fmt, dwnld_dir, verbose=True, ret_download_path=True)
Confirmed 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.
"west-yorkshire-latest.osm.pbf" is already available at "\tests".

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

>>> for dwnld_path in dwnld_paths: os.remove(dwnld_path)
```

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

**Returns** subregion information for each continent

**Return type** dict or None

**Example**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

>>> subregion_tbls = geofabrik_downloader.get_continents_subregion_tables()

>>> print(list(subregion_tbls.keys()))
['Africa',
 'Antarctica',
 'Asia',
 'Australia and Oceania',
 'Central America',
 'Europe',
```

(continues on next page)

(continued from previous page)

```
 'North America',
 'South America']
```

### 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 (case-insensitive) available on Geofabrik's free download server
- **osm_file_format** (*str*) – OSM file format; valid values include ".osm.pbf", ".shp.zip" and ".osm.bz2"
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False

**Returns** default OSM filename for the subregion_name

**Return type** str

**Examples**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

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

>>> sr_filename = geofabrik_downloader.get_default_osm_filename(
...     sr_name, file_format)

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

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

>>> sr_filename = geofabrik_downloader.get_default_osm_filename(
...     sr_name, file_format)
No .shp.zip data is available to download for Great Britain.

>>> print(sr_filename)
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 (case-insensitive) available on Geofabrik's free download server
- **osm_file_format** (*str*) – OSM file format; valid values include ".osm.pbf", ".shp.zip" and ".osm.bz2"
- **mkdir** (*bool*) – whether to create a directory, defaults to False
- **update** (*bool*) – whether to check on update and proceed to 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()

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

>>> filename, file_path = geofabrik_downloader.get_default_path_to_osm_file(
...     sr_name, file_format)

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

>>> print(os.path.relpath(file_path))
dat_GeoFabrik\Europe\Great Britain\England\greater-london-latest.osm.pbf
```

### GeofabrikDownloader.get_download_catalogue

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

Get a catalogue of download information.

Similar to *.get_download_index()*.

**Parameters**

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

**Returns** a catalogues for subregion downloads

**Return type** pandas.DataFrame or None

**Example**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

>>> downloads_catalogue = geofabrik_downloader.get_download_catalogue()

>>> print(downloads_catalogue.head())
      Subregion  ...                                            .osm.bz2
0       Algeria  ...  http://download.geofabrik.de/africa/algeria-la...
1        Angola  ...  http://download.geofabrik.de/africa/angola-lat...
2         Benin  ...  http://download.geofabrik.de/africa/benin-late...
3      Botswana  ...  http://download.geofabrik.de/africa/botswana-l...
4  Burkina Faso  ...  http://download.geofabrik.de/africa/burkina-fa...

[5 rows x 6 columns]
```

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

**Returns** the formal index of all downloads

**Return type** pandas.DataFrame or None

**Example**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

>>> download_idx = geofabrik_downloader.get_download_index()

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

**Returns** names of geographic regions available on the free download server

**Return type** list

**Example**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

>>> sr_name_list = geofabrik_downloader.get_list_of_subregion_names()

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

**Returns** a table of raw directory index

**Return type** pandas.DataFrame or None

**Examples**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

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

>>> raw_dir_idx = geofabrik_downloader.get_raw_directory_index(ex_url)

>>> print(raw_dir_idx.head())
                              File  ...                              FileURL
0            great-britain-updates  ...  https://download.geofabrik.de/...
1  great-britain-latest.osm.pbf.md5  ...  https://download.geofabrik.de/...
2  great-britain-200914.osm.pbf.md5  ...  https://download.geofabrik.de/...
3                great-britain.kml  ...  https://download.geofabrik.de/...
4     great-britain-latest.osm.pbf  ...  https://download.geofabrik.de/...

[5 rows x 4 columns]

>>> ex_url = 'http://download.geofabrik.de/'

>>> raw_dir_idx = geofabrik_downloader.get_raw_directory_index(
...     ex_url, verbose=True)
The web page does not have a raw directory index.
```

### 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 update and proceed to update the package data, defaults to False
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True

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

**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

>>> geofabrik_downloader = GeofabrikDownloader()

>>> rs_tier, ns_list = geofabrik_downloader.get_region_subregion_tier()

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

>>> print(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 (case-insensitive) available on Geofabrik's free download server
- **osm_file_format** (*str*) – OSM file format available on the free download server; valid values include `".osm.pbf"`, `".shp.zip"` and `".osm.bz2"`
- **update** (*bool*) – whether to check on update and proceed to 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()

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

>>> fml_name, dwnld_link = geofabrik_downloader.get_subregion_download_url(
...     region_name, file_format)

>>> print(fml_name)
Greater London
>>> print(dwnld_link)
http://download.geofabrik.de/.../greater-london-latest.osm.pbf

>>> region_name  = 'Great Britain'
>>> file_format = '.shp'

>>> fml_name, dwnld_link = geofabrik_downloader.get_subregion_download_url(
...     region_name, file_format)

>>> print(fml_name)
Greater London
>>> 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()

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

>>> subregion_tbl = geofabrik_downloader.get_subregion_table(ex_url)

>>> print(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]
```

### 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 (case-insensitive) available on Geofabrik's free download server
- **osm_file_format** (*str*) – OSM file format; valid values include ".osm.pbf", ".shp.zip" and ".osm.bz2"
- **download_dir** (*str or None*) – directory for saving the downloaded file(s); if None (default), the default directory
- **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()

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

>>> dwnld_dir = geofabrik_downloader.make_sub_download_dir(
...     sr_name, file_format)

>>> print(os.path.relpath(dwnld_dir))
# dat_GeoFabrik\Europe\Great Britain\England\greater-london-latest-osm-pbf

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

>>> dwnld_dir = geofabrik_downloader.make_sub_download_dir(
...     sr_name, file_format, download_dir="tests")

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

### GeofabrikDownloader.osm_file_exists

GeofabrikDownloader.**osm_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.

**Parameters**

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) available on Geofabrik's free download server
- **osm_file_format** (*str*) – OSM file format; valid values include ".osm.pbf", ".shp.zip" and ".osm.bz2"
- **data_dir** (*str or None*) – directory for saving the downloaded file(s); if None (default), use the default directory
- **update** (*bool*) – whether to check on update and proceed to update the package 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 requested data file exists

**Return type** bool

**Examples**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

>>> sr_name = 'london'
>>> file_fmt = ".pbf"

>>> path_to_pbf = geofabrik_downloader.osm_file_exists(
...     sr_name, file_fmt, verbose=True)

>>> print(path_to_pbf)
True  # (if the PBF data file exists)

>>> path_to_pbf = geofabrik_downloader.osm_file_exists(
...     sr_name, file_fmt, ret_file_path=True)

>>> print(os.path.relpath(path_to_pbf))
# (if the data file exists)
dat_GeoFabrik\Europe\Great Britain\England\greater-london-latest.osm.pbf
```

### 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 (case-insensitive) available on Geofabrik's 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()

>>> sr_names = geofabrik_downloader.search_for_subregions()
>>> print(sr_names[:5])
['Antarctica', 'Algeria', 'Angola', 'Benin', 'Botswana']

>>> sr_names = geofabrik_downloader.search_for_subregions(
...     'england', 'asia', deep=False)
>>> print(sr_names[:5])
['Bedfordshire', 'Berkshire', 'Bristol', 'Buckinghamshire', 'Cambridgeshire']
>>> print(sr_names[-5:])
['Thailand', 'Turkmenistan', 'Uzbekistan', 'Vietnam', 'Yemen']

>>> sr_names = geofabrik_downloader.search_for_subregions(
...     'britain', deep=True)
>>> print(sr_names[:5])
['Scotland', 'Wales', 'Bedfordshire', 'Berkshire', 'Bristol']
```

### 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's free download server.

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

**Returns** formal file format

**Return type** str

**Examples**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

>>> file_format = ".pbf"
>>> file_fmt = geofabrik_downloader.validate_input_file_format(file_format)

>>> print(file_fmt)
.osm.pbf

>>> file_format = ".shp"
>>> file_fmt = geofabrik_downloader.validate_input_file_format(file_format)

>>> print(file_fmt)
.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's free download server.

**Parameters** **subregion_name** (*str*) – name of a geographic region (case-insensitive) available on Geofabrik's free download server

**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()

>>> sr_name = 'london'
>>> sr_name_ = geofabrik_downloader.validate_input_subregion_name(sr_name)

>>> print(sr_name_)
Greater London

>>> sr_name = 'https://download.geofabrik.de/europe/great-britain.html'
>>> sr_name_ = geofabrik_downloader.validate_input_subregion_name(sr_name)

>>> print(sr_name_)
Great Britain
```

### 3.1.2 BBBikeDownloader

**class** pydriosm.downloader.**BBBikeDownloader**

A class for downloading OSM data from BBBike's free download server.

**Example**:

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

>>> bbbike_downloader = BBBikeDownloader()

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

#### 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_na ...]) | Download OSM data of all available formats for a geographic region. |
| *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_osm_file_formats*() | Get a list of valid OSM data file formats. |
| *get_subregion_catalogue*([update, ...]) | Get a catalogue for geographic regions. |
| *get_subregion_download_catalogue*(su | 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_na ...) | Get information of downloading (or downloaded) data file. |
| *validate_input_file_format*(osm_file_ | Validate an input file format of OSM data. |
| *validate_input_subregion_name*(subre | 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_sec=1*, *verbose=False*, *ret_download_path=False*)

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

**Parameters**

- **subregion_names** (*str or list*) – name(s) of one (or multiple) geographic region(s) available on BBBike's 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), package data directory
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **interval_sec** (*int*) – interval (in sec) between downloading two subregions, defaults to 1
- **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=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()

>>> sr_names = 'London'
>>> file_fmt = 'pbf'

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

>>> # Delete the directory generated above
```

(continues on next page)

(continued from previous page)

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

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

>>> dwnld_paths = bbbike_downloader.download_osm_data(sr_names, file_fmt,
...                                                   dwnld_dir, verbose=True,
...                                                   ret_download_path=True)
Confirmed 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*, *verbose=False*, *ret_download_path=False*)

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

**Parameters**

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) available on BBBike's free download server
- **download_dir** (*str or None*) – directory where the downloaded file is saved, defaults to None
- **update** (*bool*) – whether to check on update and proceed to update the package data, defaults to False
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to True
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to False

- **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=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()

>>> sr_name = 'london'

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

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

>>> sr_name = 'leeds'
>>> dwnld_dir = "tests"

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

(continues on next page)

(continued from previous page)

```
    Leeds.osm.csv.xz ...
    Leeds.poly ...
    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.dirname(dwnld_paths[0]), confirmation_required=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 update and proceed to update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

**Returns** location information of BBBike cities

**Return type** pandas.DataFrame or None

**Example**:

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

>>> bbbike_downloader = BBBikeDownloader()

>>> coords_of_cities = bbbike_downloader.get_coordinates_of_cities()

>>> print(coords_of_cities.tail())
          City           Real name  ... ur_longitude ur_latitude
```

(continues on next page)

(continued from previous page)

```
233     Zagreb   de!Agram,en!Zagreb  ...       16.291       45.94
234    Zuerich  de!Zürich,en!Zurich  ...         8.87       47.58
238     bbbike                      ...    14.249353   52.355108
240      dummy                      ...      44.5259     33.4238
241  Finowfurt                      ...      13.8591     52.8787

[5 rows x 13 columns]
```

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

**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()

>>> dwnld_dict = bbbike_downloader.get_download_index()

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

>>> print(dwnld_dict['Catalogue']['Leeds'].head())
                           Filename  ...          LastUpdate
0                      Leeds.osm.pbf  ... 2020-08-14 18:10:47
1                       Leeds.osm.gz  ... 2020-08-14 23:26:15
2                   Leeds.osm.shp.zip  ... 2020-08-14 23:48:29
3  Leeds.osm.garmin-onroad-latin1.zip  ... 2020-08-15 01:59:13
4          Leeds.osm.garmin-onroad.zip  ... 2020-08-15 01:59:02

[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.

**Parameters**

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

**Returns** catalogue for subregions of BBBike data

**Return type** pandas.DataFrame or None

**Example**:

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

>>> bbbike_downloader = BBBikeDownloader()

>>> names_of_cities = bbbike_downloader.get_list_of_cities()

>>> print(names_of_cities[:5])
['Heilbronn', 'Emden', 'Bremerhaven', 'Paris', 'Ostrava']
```

### 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.

**Parameters**

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

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

**Return type** list

**Example**:

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

>>> bbbike_downloader = BBBikeDownloader()

>>> sr_name_list = bbbike_downloader.get_list_of_subregion_names()

>>> print(sr_name_list[:5])
['Aachen', 'Aarhus', 'Adelaide', 'Albuquerque', 'Alexandria']
```

### BBBikeDownloader.get_osm_file_formats

BBBikeDownloader.**get_osm_file_formats**()

Get a list of valid OSM data file formats.

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

**Return type** list

**Example**:

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

>>> bbbike_downloader = BBBikeDownloader()

>>> file_fmts = bbbike_downloader.get_osm_file_formats()

>>> for file_fmt in file_fmts:
...     print(file_fmt)
.pbf
.gz
.shp.zip
.garmin-onroad-latin1.zip
.garmin-onroad.zip
.garmin-opentopo.zip
.garmin-osm.zip
.geojson.xz
.svg-osm.zip
.mapsforge-osm.zip
.navit.zip
.csv.xz
```

### 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 update and proceed to update the package data, defaults to `False`
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`

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

**Returns** catalogue for subregions of BBBike data

**Return type** pandas.DataFrame or None

**Example**:

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

>>> bbbike_downloader = BBBikeDownloader()

>>> subregion_catalog = bbbike_downloader.get_subregion_catalogue()

>>> print(subregion_catalog.head())
          Name  ...                                                URL
1       Aachen  ...      http://download.bbbike.org/osm/bbbike/Aachen/
2       Aarhus  ...      http://download.bbbike.org/osm/bbbike/Aarhus/
3     Adelaide  ...    http://download.bbbike.org/osm/bbbike/Adelaide/
4  Albuquerque  ...  http://download.bbbike.org/osm/bbbike/Albuquer...
5   Alexandria  ...  http://download.bbbike.org/osm/bbbike/Alexandria/

[5 rows x 3 columns]
```

### 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 (case-insensitive) available on BBBike's free download server
- **confirmation_required** (*bool*) – whether to prompt a message for confirmation to proceed, defaults to `True`
- **verbose** (*bool or int*) – whether to print relevant information in console, defaults to `False`

**Returns** a catalogues for subregion downloads

**Return type** pandas.DataFrame or None

**Example**:

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

>>> bbbike_downloader = BBBikeDownloader()

>>> sr_name = 'leeds'

>>> leeds_dwnld_cat = bbbike_downloader.get_subregion_download_catalogue(
...     subregion_name=sr_name, verbose=True)
Confirmed to collect the download catalogue for Leeds? [No]|Yes: yes
In progress ... Done.
```

(continues on next page)

(continued from previous page)

```
>>> print(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]
```

### 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 (case-insensitive) available on BBBike's 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()

>>> sr_name = 'leeds'
>>> file_fmt = 'pbf'

>>> sr_name_, sr_url = bbbike_downloader.get_subregion_download_url(
...     sr_name, file_fmt)

>>> print(sr_name_)
Leeds
>>> print(sr_url)
http://download.bbbike.org/osm/bbbike/Leeds/Leeds.osm.pbf

>>> file_fmt = 'csv.xz'
>>> sr_name_, sr_url = bbbike_downloader.get_subregion_download_url(
...     sr_name, file_fmt)

>>> print(sr_name_)
Leeds
>>> print(sr_url)
http://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*)

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), package data directory

**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()

>>> sr_name = 'leeds'
>>> file_fmt = 'pbf'

>>> info = bbbike_downloader.get_valid_download_info(sr_name, file_fmt)
>>> sr_name_, pbf_filename, dwnld_url, path_to_pbf = info

>>> print(sr_name_)
Leeds
>>> print(pbf_filename)
Leeds.osm.pbf
>>> print(dwnld_url)
http://download.bbbike.org/osm/bbbike/Leeds/Leeds.osm.pbf
>>> print(os.path.relpath(path_to_pbf))
dat_BBBike\Leeds\Leeds.osm.pbf
```

### 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's 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_fmt = 'PBF'

>>> file_fmt_ = bbbike_downloader.validate_input_file_format(file_fmt)

>>> print(file_fmt_)
.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's 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()

>>> sr_name = 'leeds'

>>> sr_name_ = bbbike_downloader.validate_input_subregion_name(sr_name)

>>> print(sr_name_)
Leeds
```

## 3.2 reader

Reading OSM data extracts.

### Classes

| | |
|---|---|
| *GeofabrikReader*([max_tmpfile_size]) | A class representation of a tool for reading Geofabrik data extracts. |
| *BBBikeReader*([max_tmpfile_size]) | A class representation of a tool for reading BBBike data extracts. |

## 3.2.1 GeofabrikReader

**class** pydriosm.reader.**GeofabrikReader**(*max_tmpfile_size=5000*)

A class representation of a tool for reading Geofabrik data extracts.

**Parameters** **max_tmpfile_size** (*int or None*) – defaults to 5000, see also *pydriosm.settings.gdal_configurations()*

**Example**:

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

>>> geofabrik_reader = GeofabrikReader()

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

### Methods

| | |
|---|---|
| *get_path_to_osm_pbf*(subregion_name[, data_dir]) | Get the absolute local path to a PBF (.osm.pbf) data file for a geographic region. |
| *get_path_to_osm_shp*(subregion_name[, ...]) | Get the absolute path(s) to .shp file(s) for a geographic region (by searching a local data directory). |
| *merge_subregion_layer_shp*(layer_nam ...[, ...]) | 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_path_to_osm_pbf

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

Get the absolute local path to a PBF (.osm.pbf) data file for a geographic region.

**Parameters**

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) available on Geofabrik's 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()

>>> sr_name = 'Rutland'

>>> path_to_rutland_pbf = geofabrik_reader.get_path_to_osm_pbf(sr_name)

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

>>> file_fmt = ".pbf"
>>> dwnld_dir = "tests"

>>> # Download the PBF data file of Rutland to "\tests"
>>> geofabrik_reader.Downloader.download_osm_data(sr_name, file_fmt,
...                                               dwnld_dir, verbose=True)
Confirmed 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_pbf(
...     sr_name, 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 the absolute path(s) to .shp file(s) for a geographic region (by searching a local data directory).

**Parameters**

- **subregion_name** (*str*) – name of a region/subregion (case-insensitive) available on Geofabrik's 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
>>> from pydriosm.reader import unzip_shp_zip, parse_layer_shp

>>> geofabrik_reader = GeofabrikReader()

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

>>> path_to_shp_file = geofabrik_reader.get_path_to_osm_shp(sr_name)
>>> print(path_to_shp_file)
# (if "gis.osm_railways_free_1.shp" is unavailable at the package data␣
 ↪directory)
[]

>>> dwnld_dir = "tests"

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

>>> unzip_shp_zip(path_to_rutland_shp_zip, verbose=True)
Extracting all ... to "\tests\rutland-latest-free-shp" ...
In progress ... Done.

>>> lyr_name = 'railways'

>>> path_to_rutland_railways_shp = geofabrik_reader.get_path_to_osm_shp(
```

(continues on next page)

(continued from previous page)

```
...     sr_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'

>>> _ = parse_layer_shp(path_to_rutland_railways_shp, feature_names=feat_name,
...                     save_fclass_shp=True)

>>> path_to_rutland_railways_rail_shp = geofabrik_reader.get_path_to_osm_shp(
...     sr_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

>>> # 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**(*layer_name*, *subregion_names*, *data_dir=None*, *method='geopandas'*, *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*) – a list of region/subregion names (case-insensitive) that are available on Geofabrik's free download server
- **layer_name** (*str*) – name of a layer (e.g. 'railways')
- **method** (*str*) – the method used to merge/save .shp files; if `'geopandas'` (default), use the geopandas.GeoDataFrame.to_file method, otherwise, use 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`, 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 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

**Examples**:

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

>>> geofabrik_reader = GeofabrikReader()

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

>>> path_to_merged_shp_file = geofabrik_reader.merge_subregion_layer_shp(
...     lyr_name, sr_names, dat_dir, verbose=True, ret_merged_shp_path=True)
Confirmed to download .shp.zip data of the following geographic region(s):
    Greater Manchester
    West Yorkshire
? [No]|Yes: yes
Downloading "greater-manchester-latest-free.shp.zip" to "\tests" ...
Done.
Downloading "west-yorkshire-latest-free.shp.zip" to "\tests" ...
Done.
Extracting from "greater-manchester-latest-free.shp.zip" the following layer(s):
    'railways'
to "\tests\greater-manchester-latest-free-shp" ...
In progress ... Done.
Extracting from "west-yorkshire-latest-free.shp.zip" the following layer(s):
    'railways'
to "\tests\west-yorkshire-latest-free-shp" ...
In progress ... Done.
Merging the following shapefiles:
    "greater-manchester_gis_osm_railways_free_1.shp"
    "west-yorkshire_gis_osm_railways_free_1.shp"
In progress ... Done.
```

(continues on next page)

(continued from previous page)

```
Find ... file(s) at "\tests\greater-manchester_west-yorkshire_railways".

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

>>> # 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"))

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

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

>>> path_to_merged_shp_files = geofabrik_reader.merge_subregion_layer_shp(
...     lyr_name, sr_names, dat_dir, verbose=True, ret_merged_shp_path=True)
Confirmed 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 from "greater-london-latest-free.shp.zip" the following layer(s):
    'transport'
to "\tests\greater-london-latest-free-shp" ...
In progress ... Done.
Extracting from "kent-latest-free.shp.zip" the following layer(s):
    'transport'
to "\tests\kent-latest-free-shp" ...
In progress ... Done.
Extracting from "surrey-latest-free.shp.zip" the following layer(s):
    'transport'
to "\tests\surrey-latest-free-shp" ...
In progress ... 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".

>>> for path_to_merged_shp_file in path_to_merged_shp_files:
...     print(os.path.relpath(path_to_merged_shp_file))
tests\...\greater-london_kent_surrey_transport_point.shp
tests\...\greater-london_kent_surrey_transport_polygon.shp
```

(continues on next page)

(continued from previous page)

```
>>> # Delete the merged files
>>> delete_dir(os.path.commonpath(path_to_merged_shp_files), 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*)

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

**Parameters**

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) available on Geofabrik's 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*) – whether to return an absolute path to the saved pickle file (when `pickle_it=True`)
- **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`

**Returns** dictionary of the .osm.pbf data; when `pickle_it=True`, return a tuple of the dictionary and an absolute 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"

>>> rutland_pbf_raw = geofabrik_reader.read_osm_pbf(sr_name, dat_dir,
...                                                 verbose=True)
Confirmed to download .osm.pbf data of the following geographic region(s):
    Rutland
? [No]|Yes: yes

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

>>> rutland_pbf_raw_points = rutland_pbf_raw['points']
>>> print(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...

>>> 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']
>>> print(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]

>>> rutland_pbf_parsed_1 = geofabrik_reader.read_osm_pbf(sr_name, dat_dir,
...                                                     parse_raw_feat=True,
...                                                     transform_geom=True,
```

(continues on next page)

(continued from previous page)

```
...                                                    verbose=True)
Parsing "\tests\rutland-latest.osm.pbf" ... Done.

>>> rutland_pbf_parsed_1_points = rutland_pbf_parsed_1['points']
>>> print(rutland_pbf_parsed_1_points[['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)

>>> 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)

>>> rutland_pbf_parsed_2_points = rutland_pbf_parsed_2['points']
>>> print(rutland_pbf_parsed_2_points[['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(f"{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 region/subregion (case-insensitive) available on Geofabrik's 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*) – whether to return an absolute path to the saved pickle file (when `pickle_it=True`)
- **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(sr_name, data_dir=dat_dir)
Confirmed to download .shp.zip data of the following geographic region(s):
    Rutland
? [No]|Yes: yes

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

>>> rutland_shp_railways = rutland_shp['railways']
>>> print(rutland_shp_railways.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...
```

(continues on next page)

(continued from previous page)

```
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]

>>> 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.

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

>>> print(rutland_shp_transport['transport'].head())
      osm_id  code    fclass                    name                    geometry
0  472398147  5621  bus_stop                    None  POINT (-0.73213 52.66974)
1  502322073  5621  bus_stop             Fife Close  POINT (-0.50962 52.66052)
2  502322075  5621  bus_stop             Fife Close  POINT (-0.50973 52.66058)
3  502322076  5621  bus_stop          Aberdeen Close  POINT (-0.51039 52.65817)
4  502322077  5621  bus_stop  Arran Road (South End)  POINT (-0.50973 52.65469)

>>> feat_name = 'bus_stop'

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

>>> print(list(rutland_shp_transport_bus_stop.keys()))
['transport']

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

>>> 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 from "rutland-latest-free.shp.zip" the following layer(s):
    'traffic'
    'roads'
to "\tests\rutland-latest-free-shp" ...
In progress ... Done.
Deleting the extracts "\tests\rutland-latest-free-shp"  ... Done.
Deleting "tests\rutland-latest-free.shp.zip" ... Done.

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

>>> selected_columns = ['fclass', 'name', 'geometry']

>>> rutland_shp_tr_pt_traffic = rutland_shp_tr_pt['traffic']
>>> print(rutland_shp_tr_pt_traffic[selected_columns].head())
```

(continues on next page)

(continued from previous page)

```
    fclass  name                                              geometry
0  parking  None  POLYGON ((-0.66704 52.71108, -0.66670 52.71121...
1  parking  None  POLYGON ((-0.78712 52.71974, -0.78700 52.71991...
2  parking  None  POLYGON ((-0.70368 52.65567, -0.70362 52.65587...
3  parking  None  POLYGON ((-0.63381 52.66442, -0.63367 52.66441...
4  parking  None  POLYGON ((-0.62814 52.64093, -0.62701 52.64169...

>>> rutland_shp_tr_pt_roads = rutland_shp_tr_pt['roads']
>>> print(rutland_shp_tr_pt_roads[selected_columns].head())
   fclass           name                                              geometry
0   trunk           None  LINESTRING (-0.72461 52.59642, -0.72452 52.596...
1   trunk   Glaston Road  LINESTRING (-0.64671 52.59353, -0.64590 52.593...
3   trunk  Orange Street  LINESTRING (-0.72293 52.58899, -0.72297 52.588...
11  trunk    Ayston Road  LINESTRING (-0.72483 52.59610, -0.72493 52.596...
12  trunk    London Road  LINESTRING (-0.72261 52.58759, -0.72264 52.587...
```

### 3.2.2 BBBikeReader

**class** pydriosm.reader.**BBBikeReader**(*max_tmpfile_size=5000*)

A class representation of a tool for reading BBBike data extracts.

**Parameters** **max_tmpfile_size** (*int or None*) – defaults to 5000, see also *pydriosm.settings.gdal_configurations()*

**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 absolute 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 absolute 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) available on BBBike's 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()

>>> sr_name = 'Leeds'
>>> file_fmt = ".pbf"
>>> dat_dir = "tests"

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

>>> path_to_leeds_pbf_ = bbbike_reader.get_path_to_osm_file(
...     sr_name, file_fmt, dat_dir)
>>> print(os.path.relpath(path_to_leeds_pbf_))
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) available on BBBike's 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()

>>> sr_name = 'Leeds'
>>> dat_dir = "tests"

>>> leeds_csv_xz = bbbike_reader.read_csv_xz(sr_name, dat_dir, verbose=True)
Confirmed 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.

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

>>> # 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 region/subregion (case-insensitive) available on BBBike's 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()

>>> sr_name = 'Leeds'
>>> dat_dir = "tests"

>>> leeds_geojson_xz = bbbike_reader.read_geojson_xz(sr_name, dat_dir,
...                                                  verbose=True)
Confirmed 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.

>>> print(leeds_geojson_xz.head())
  feature_name  ...                                         properties
0      Feature  ...  {'ref': '40', 'name': 'Flushdyke', 'highway': ...
1      Feature  ...  {'ref': '44', 'name': 'Bramham', 'highway': 'm...
2      Feature  ...  {'ref': '43', 'name': 'Belle Isle', 'highway':...
3      Feature  ...  {'ref': '42', 'name': 'Lofthouse', 'highway': ...
4      Feature  ...  {'ref': '42', 'name': 'Lofthouse', 'highway': ...
[5 rows x 4 columns]

>>> print(leeds_geojson_xz[['coordinates']].head())
                coordinates
```

(continues on next page)

(continued from previous page)

```
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]

>>> leeds_geojson_xz_ = bbbike_reader.read_geojson_xz(sr_name, dat_dir,
...                                                   fmt_geom=True)

>>> print(leeds_geojson_xz_[['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*)

Read a PBF data file of a geographic region.

**Parameters**

- **subregion_name** (*str*) – name of a geographic region (case-insensitive) available on BBBike's 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*) – whether to return an absolute path to the saved pickle file (when `pickle_it=True`)
- **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`

**Returns** dictionary of the .osm.pbf data; when `pickle_it=True`, return a tuple of the dictionary and an absolute 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()

>>> sr_name = 'Leeds'
>>> dat_dir = "tests"

>>> # (Note that this process may take a long time.)
>>> leeds_osm_pbf = bbbike_reader.read_osm_pbf(sr_name, dat_dir,
...                                            parse_raw_feat=True,
...                                            transform_geom=True,
...                                            transform_other_tags=True,
...                                            verbose=True)
Parsing "\tests\Leeds.osm.pbf" ... Done.

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

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

>>> # Delete the downloaded PBF data file
>>> os.remove(cd(data_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) available on BBBike's 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*) – whether to return an absolute path to the saved pickle file (when `pickle_it=True`)
- **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 an absolute path to the pickle file

**Return type** dict or tuple or None

**Examples**:

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

>>> bbbike_reader = BBBikeReader()

>>> sr_name = 'Birmingham'
>>> dat_dir = "tests"

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

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

>>> birmingham_railways_shp = birmingham_shp['railways']
>>> print(birmingham_railways_shp.head())
    osm_id  ...                                           geometry
0      740  ...  LINESTRING (-1.81789 52.57010, -1.81793 52.569...
1     2148  ...  LINESTRING (-1.87319 52.50555, -1.87271 52.505...
2  2950000  ...  LINESTRING (-1.87941 52.48138, -1.87960 52.481...
3  3491845  ...  LINESTRING (-1.74060 52.51858, -1.73942 52.518...
4  3981454  ...  LINESTRING (-1.77475 52.52284, -1.77449 52.522...
[5 rows x 4 columns]

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

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


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

>>> print(birmingham_roads_shp['roads'].head())
   osm_id  ...                                           geometry
0      37  ...  LINESTRING (-1.82675 52.55580, -1.82646 52.555...
1      38  ...  LINESTRING (-1.81541 52.54785, -1.81475 52.547...
2      41  ...  LINESTRING (-1.81931 52.55219, -1.81860 52.552...
3      42  ...  LINESTRING (-1.82492 52.55504, -1.82309 52.556...
4      45  ...  LINESTRING (-1.82121 52.55389, -1.82056 52.55432)
```

(continues on next page)

(continued from previous page)

```
[5 rows x 8 columns]

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

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

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

>>> bham_rw_rc_shp_railways = bham_rw_rc_shp['railways']
>>> print(bham_rw_rc_shp_railways[['fclass', 'name']].head())
  fclass                            name
0   rail                 Cross-City Line
1   rail                 Cross-City Line
2   rail                            None
3   rail  Birmingham to Peterborough Line
4   rail                    Freight Line

>>> bham_rw_rc_shp_waterways = bham_rw_rc_shp['waterways']
>>> print(bham_rw_rc_shp_waterways[['fclass', 'name']].head())
   fclass                                          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
```

### Functions

| | |
|---|---|
| *get_osm_pbf_layer_names*(path_to_osm_pl | Get 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. |
| *unzip_shp_zip*(path_to_shp_zip[, ...]) | Unzip a .shp.zip file data. |
| *read_shp_file*(path_to_shp[, method]) | Parse a shapefile. |
| *get_default_shp_crs*() | Get default CRS for saving shapefile format data. |
| *parse_layer_shp*(path_to_layer_shp[, ...]) | Parse a layer of OSM shapefile data. |
| *merge_shps*(paths_to_shp_files, ... [, method]) | Merge multiple shapefiles. |

continues on next page

| Table 8 – continued from previous page | |
|---|---|
| *merge_layer_shps*(paths_to_shp_zip_files, ...) | Merge shapefiles over a layer for multiple geographic regions. |
| *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. |

### 3.2.3 get_osm_pbf_layer_names

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

Get names of all layers in a PBF data file.

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

**Returns** name (and index) of each layer of the PBF data file

**Return type** dict

**Example**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

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

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

>>> 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*, *number_of_chunks*, *parse_raw_feat*, *transform_geom*, *transform_other_tags*, *max_tmpfile_size=None*)

Parse a PBF data file.

**Parameters**

- **path_to_osm_pbf** (*str*) – absolute path to a PBF data file
- **number_of_chunks** (*int or None*) – number of chunks
- **parse_raw_feat** (*bool*) – whether to parse each feature in the raw data
- **transform_geom** – 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
- **max_tmpfile_size** (*int or None*) – defaults to `None`, see also *pydriosm.settings.gdal_configurations()*

**Returns** parsed OSM PBF data

**Return type** dict

---

**Note:** This function can require fairly high amount of physical memory to read large files e.g. > 200MB

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].

---

**Example**:

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

>>> geofabrik_downloader = GeofabrikDownloader()

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

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

>>> rutland_pbf_raw = parse_osm_pbf(path_to_rutland_pbf, number_of_chunks=50,
...                                 parse_raw_feat=False, transform_geom=False,
...                                 transform_other_tags=False)

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

>>> rutland_pbf_raw_points = rutland_pbf_raw['points']
>>> print(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...

>>> rutland_pbf_parsed = parse_osm_pbf(path_to_rutland_pbf, number_of_chunks=50,
...                                    parse_raw_feat=True, transform_geom=False,
...                                    transform_other_tags=False)

>>> rutland_pbf_parsed_points = rutland_pbf_parsed['points']
>>> print(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
```

(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]

>>> rutland_pbf_parsed_1 = parse_osm_pbf(path_to_rutland_pbf, number_of_chunks=50,
...                                      parse_raw_feat=True, transform_geom=True,
...                                      transform_other_tags=False)

>>> rutland_pbf_parsed_points_1 = rutland_pbf_parsed_1['points']
>>> print(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)

>>> rutland_pbf_parsed_2 = parse_osm_pbf(path_to_rutland_pbf, number_of_chunks=50,
...                                      parse_raw_feat=True, transform_geom=True,
...                                      transform_other_tags=True)

>>> rutland_pbf_parsed_points_2 = rutland_pbf_parsed_2['points']
>>> print(rutland_pbf_parsed_points_2[['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 (-0.7249922 52.6748223)  {'traffic_calming': 'cushion'}
4          POINT (-0.7266686 52.6695051)      {'direction': 'clockwise'}

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

**See also:**

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

### 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 data.

**Parameters**

- **path_to_shp_zip** (*str*) – absolute path to a zipped shapefile data (.shp.zip)
- **path_to_extract_dir** (*str or None*) – absolute path to a directory where extracted files will be saved; if `None` (default), use the same directory where the .shp.zip file is
- **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=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()

>>> 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, ret_download_path=True)
Confirmed to download .shp.zip data of the following geographic region(s):
    Rutland
? [No]|Yes: yes

>>> layer_name = 'railways'

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

>>> path_to_rutland_shp_dir = unzip_shp_zip(path_to_rutland_shp_zip, verbose=True,
...                                         ret_extract_dir=True)
Extracting all of "rutland-latest-free.shp.zip" to "\tests\rutland-latest-free-shp"
In progress ... 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 from "rutland-latest-free.shp.zip" the following layer(s):
    'railways'
    'transport'
    'traffic'
to "\tests\rutland-latest-free-shp" ...
In progress ... Done.
Clustering the layer data ...
```

(continues on next page)

(continued from previous page)

```
    railways ...
    transport ...
    traffic ...
    traffic_a ...
    transport_a ...
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.dirname(path_to_lyr_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 .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='geopandas'*, ***kwargs*)

Parse a shapefile.

**Parameters**

- **path_to_shp** – absolute path to a .shp data file
- **method** (*str*) – the method used to read the .shp file; if `'geopandas'` (default), use the geopandas.read_file() method, for otherwise use shapefile.Reader()
- **kwargs** – optional parameters of geopandas.read_file()

**Type** str

**Returns** data frame of the .shp data

**Return type** pandas.DataFrame or geopandas.GeoDataFrame

**Examples**:

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

>>> 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, ret_download_path=True)
Confirmed to download .shp.zip data of the following geographic region(s):
    Rutland
```

(continues on next page)

(continued from previous page)

```
? [No]|Yes: yes

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

>>> railways_shp_filename = "gis_osm_railways_free_1.shp"
>>> path_to_rutland_railways_shp = cd(
...     path_to_rutland_shp_dir, railways_shp_filename)

>>> rutland_railways_shp = read_shp_file(path_to_rutland_railways_shp,
...                                      method='gpd')

>>> print(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]

>>> rutland_railways_shp_ = read_shp_file(path_to_rutland_railways_shp,
...                                       method='pyshp')

>>> print(rutland_railways_shp_.head())
    osm_id  code  ...                                          coords 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_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_default_shp_crs

pydriosm.reader.**get_default_shp_crs**()

Get default CRS for saving shapefile format data.

**Returns** default settings of CRS

**Return type** dict

**Example**:

```
>>> from pydriosm.reader import get_default_shp_crs

>>> default_shp_crs = get_default_shp_crs()

>>> print(default_shp_crs)
{'no_defs': True, 'ellps': 'WGS84', 'datum': 'WGS84', 'proj': 'longlat'}
```

### 3.2.9 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*) – absolute 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.reader import GeofabrikDownloader
>>> from pydriosm.reader import parse_layer_shp, unzip_shp_zip

>>> 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")

>>> rutland_railways_shp = parse_layer_shp(path_to_railways_shp)

>>> print(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...
```

(continues on next page)

(continued from previous page)

```
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]

>>> 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)

>>> print(rutland_railways_rail.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]

>>> 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)
```

### 3.2.10 merge_shps

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

Merge multiple shapefiles.

**Parameters**

- **paths_to_shp_files** (*list*) – list of absolute paths to shapefiles (in .shp format)
- **path_to_merged_dir** (*str*) – absolute path to a directory where the merged files are to be saved
- **method** (*str*) – the method used to merge/save .shp files; if `'geopandas'` (default), use the geopandas.GeoDataFrame.to_file method, use shapefile.Writer otherwise

See the example for the function *merge_layer_shps()*.

### 3.2.11 merge_layer_shps

pydriosm.reader.**merge_layer_shps**(*paths_to_shp_zip_files*, *layer_name*, *method='geopandas'*, *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 absolute 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 .shp files; if `'geopandas'` (default), use the geopandas.GeoDataFrame.to_file method, use shapefile.Writer otherwise
- **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

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

>>> geofabrik_downloader = GeofabrikDownloader()

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

(continues on next page)

(continued from previous page)

```
>>> 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)

>>> 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 from "greater-manchester-latest-free.shp.zip" the following layer(s):
    'railways'
to "\tests\greater-manchester-latest-free-shp" ...
In progress ... Done.
Extracting from "west-yorkshire-latest-free.shp.zip" the following layer(s):
    'railways'
to "\tests\west-yorkshire-latest-free-shp" ...
In progress ... 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 .shp file(s) at "\tests\greater-manchester_west-yorkshire_railways".

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

>>> # 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()*.

### 3.2.12 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*) – absolute 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()*.

### 3.2.13 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*) – absolute 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, ...]) | A class representation of a tool for 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'*, *confirm_new_db=False*, *data_source='Geofabrik'*, *verbose=True*)

A class representation of a tool for 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 , 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'`

- **confirm_new_db** (*bool*) – whether to impose a confirmation to create a new database, defaults to `False`
- **data_source** (*str*) – source of data extracts, including `'Geofabrik'` and `'BBBike'`, defaults to `'Geofabrik'`
- **verbose** (*bool*) – whether to print relevant information in console as the function runs, defaults to `True`

**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.DataSource)
Geofabrik
>>> print(osmdb_test.Downloader)
<pydriosm.downloader.GeofabrikDownloader object at ...>

>>> # Change the data source:
>>> osmdb_test.DataSource = 'BBBike'
>>> print(osmdb_test.Downloader)
<pydriosm.downloader.BBBikeDownloader object at ...>
```

### Methods

| | |
|---|---|
| *drop_subregion_table*(subregion_table_ | Delete all or specific schemas/layers of subregion data from the database being connected. |
| *fetch_osm_data*(subregion_name[, ...]) | Fetch OSM data (of one or multiple layers) of a geographic region. |
| *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*(subreg | Get the default table name for a specific geographic region. |
| *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_1 ...]) | Import data of geographic region(s) that do not have (sub-)subregions into a database. |
| *subregion_table_exists*(subregion_nai ...) | Check if a table (for a geographic region) exists. |

### 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*) – 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), use the default layer names as schema names
- **table_named_as_subregion** (*bool*) – whether to use subregion name to be 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 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 pydriosm.ios import PostgresOSM

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

>>> # With all the examples for the methods:
>>> # `.import_osm_data()` and `.import_subregion_osm_pbf()`,
>>> # delete all data of Rutland and Leeds

>>> subregion_tbl_names = ['Rutland', 'Leeds']

>>> osmdb_test.drop_subregion_table(subregion_tbl_names, verbose=True)
Confirmed to drop the following tables:
    "Leeds" and
    "Rutland"
  from the following schemas:
    "multipolygons",
    "water",
    "multilinestrings",
    "points",
    "buildings",
    "natural",
    "roads",
    "other_relations",
    "pois",
```

(continues on next page)

(continued from previous page)

```
    "traffic",
    "transport",
    "pofw",
    "landuse",
    "railways",
    "waterways",
    "lines" and
    "places"
  at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Dropping tables ...
    "multipolygons"."Rutland" ... Done.
    "water"."Rutland" ... Done.
    "multilinestrings"."Rutland" ... Done.
    "points"."Leeds" ... Done.
    "points"."Rutland" ... Done.
    "buildings"."Leeds" ... Done.
    "buildings"."Rutland" ... Done.
    "natural"."Leeds" ... Done.
    "natural"."Rutland" ... Done.
    "roads"."Leeds" ... Done.
    "roads"."Rutland" ... Done.
    "other_relations"."Rutland" ... Done.
    "pois"."Rutland" ... Done.
    "traffic"."Rutland" ... Done.
    "transport"."Rutland" ... Done.
    "pofw"."Rutland" ... Done.
    "landuse"."Leeds" ... Done.
    "landuse"."Rutland" ... Done.
    "railways"."Leeds" ... Done.
    "railways"."Rutland" ... Done.
    "waterways"."Leeds" ... Done.
    "waterways"."Rutland" ... Done.
    "lines"."Rutland" ... Done.
    "places"."Leeds" ... Done.
    "places"."Rutland" ... 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)
Confirmed to drop the following tables:
    "Waterloo" and
    "Victoria"
  from the following schemas:
    "points" and
    "other_relations"
  at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Dropping tables ...
    "points"."Victoria" ... Done.
    "points"."Waterloo" ... Done.
    "other_relations"."Victoria" ... Done.
    "other_relations"."Waterloo" ... Done.

>>> # Delete the database 'osmdb_test'
```

(continues on next page)

(continued from previous page)

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

### 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='spooled_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), use the default layer names as schema names
- **table_named_as_subregion** (*bool*) – whether to use subregion name to be 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*, *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 'spooled_tempfile'
- **max_size_spooled** (*int*, *float*) – max_size_spooled of pyhelpers.sql.PostgreSQL.read_sql_query, defaults to 1 (in gigabyte)
- **decode_geojson** (*bool*) – whether to decode textual GeoJSON, defaults to False
- **decode_wkt** (*bool*) – whether to decode 'coordinates' (if available and) if it is a 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.

>>> sr_name = 'Rutland'

>>> # With all the examples for
>>> # `.import_osm_data()` and  `.import_subregion_osm_pbf()`,
>>> # fetch data of all available layers
>>> rutland_pbf = osmdb_test.fetch_osm_data(sr_name,
...                                         table_named_as_subregion=True)

>>> type(rutland_pbf)
<class 'dict'>
>>> print(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)
<class 'dict'>
>>> print(list(rutland_pbf.keys()))
# ['points', 'multipolygons']

>>> rutland_pbf_points = rutland_pbf['points']
>>> print(rutland_pbf_points.head())
                                                points
0  {"type": "Feature", "geometry": {"type": "Poin...
1  {"type": "Feature", "geometry": {"type": "Poin...
2  {"type": "Feature", "geometry": {"type": "Poin...
```

(continues on next page)

(continued from previous page)

```
3  {"type": "Feature", "geometry": {"type": "Poin...
4  {"type": "Feature", "geometry": {"type": "Poin...

>>> rutland_pbf_ = osmdb_test.fetch_osm_data(
...     sr_name, lyr_names, decode_geojson=True, decode_wkt=True,
...     decode_other_tags=True, sort_by='id')

>>> rutland_pbf_points_ = rutland_pbf_['points']
>>> print(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]
```

**See also:**

The examples about *fetching data from the database* provided in *Quick start*.

### 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): ***
```

(continues on next page)

(continued from previous page)

```
Connecting postgres:***@localhost:5432/osmdb_test ... Successfully.

>>> sr_name = 'rutland'
>>> lyr_name = 'points'

>>> column_info_table = osmdb_test.get_subregion_table_column_info(
...     sr_name, lyr_name)

>>> type(column_info_table)
<class 'pandas.core.frame.DataFrame'>
>>> print(column_info_table.index.to_list()[:5])
['table_catalog',
 'table_schema',
 'table_name',
 'column_name',
 'ordinal_position']

>>> column_info_dict = osmdb_test.get_subregion_table_column_info(
...     sr_name, lyr_name, as_dict=True, table_named_as_subregion=True,
...     schema_named_as_layer=True)

>>> type(column_info_dict)
<class 'dict'>
>>> print(list(column_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.
```

(continues on next page)

(continued from previous page)

```
>>> sr_name = 'rutland'

>>> tbl_name = osmdb_test.get_table_name_for_subregion(sr_name)

>>> print(tbl_name)
rutland

>>> tbl_name = osmdb_test.get_table_name_for_subregion(
...     sr_name, table_named_as_subregion=True)

>>> print(tbl_name)
# Rutland
```

**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_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 None*) – names of schemas for each layer of the PBF data, if `None` (default), use the default layer names as schema names
- **table_named_as_subregion** (*bool*) – whether to use subregion name to be 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*, *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'
>>> dat_dir = "tests"

>>> rutland_pbf_raw = osmdb_test.Reader.read_osm_pbf(sr_name, dat_dir,
...                                                  verbose=True)
Confirmed to download .osm.pbf data of the following geographic region(s):
    Rutland
? [No]|Yes: yes

>>> # Import all layers of the raw PBF data of Rutland

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

>>> rutland_pbf = osmdb_test.Reader.read_osm_pbf(sr_name, dat_dir,
...                                              parse_raw_feat=True,
...                                              transform_geom=True,
...                                              transform_other_tags=True)

>>> # Import data into specific schemas
>>> schemas = {'schema_0': 'lines',
...            'schema_1': 'points',
...            'schema_2': 'multipolygons'}

>>> osmdb_test.import_osm_data(rutland_pbf, table_name=sr_name,
...                            schema_names=schemas, verbose=True)
Confirmed to import the data into table "Rutland"
    at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing data into "Rutland" ...
    schema_0 ... done: <total of rows> features.
    schema_1 ... done: <total of rows> features.
    schema_2 ... done: <total of rows> features.
```

(continues on next page)

(continued from previous page)

```
>>> # To drop the schemas starting with 'schema_'
>>> osmdb_test.PostgreSQL.drop_schema(schemas.keys(), verbose=True)
Confirmed to drop the following schemas:
    "schema_0",
    "schema_1" and
    "schema_2"
  from postgres:***@localhost:5432/osmdb_test
? [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"))

>>> rutland_shp = osmdb_test.Reader.read_shp_zip(sr_name, data_dir=dat_dir,
...                                              rm_extracts=True,
...                                              rm_shp_zip=True,
...                                              verbose=True)
Confirmed 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 all of "rutland-latest-free.shp.zip" to "\tests\rutl-...-shp" ...
In progress ... Done.
Deleting the extracts "\tests\rutland-latest-free-shp"  ... Done.
Deleting "tests\rutland-latest-free.shp.zip" ... Done.

>>> # Import all layers of the shapefile data of Rutland

>>> osmdb_test.import_osm_data(rutland_shp, table_name=sr_name, verbose=True)
Confirmed to import the data into table "Rutland"
    at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing data into "Rutland" ...
    pofw ... done: <total of rows> features.
    pois ... done: <total of rows> features.
    traffic ... done: <total of rows> features.
    places ... done: <total of rows> features.
    waterways ... done: <total of rows> features.
    buildings ... done: <total of rows> features.
    transport ... done: <total of rows> features.
    natural ... done: <total of rows> features.
    roads ... done: <total of rows> features.
    water ... done: <total of rows> features.
    railways ... done: <total of rows> features.
    landuse ... done: <total of rows> features.

>>> # Import BBBike shapefile data

>>> 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)
Confirmed to download .shp.zip data of the following geographic region(s):
```

(continues on next page)

(continued from previous page)

```
    Leeds
? [No]|Yes: yes
Downloading "Leeds.osm.shp.zip" to "\tests" ...
Done.
Extracting all of "Leeds.osm.shp.zip" to "\tests" ...
In progress ... Done.
Parsing "\tests\Leeds-shp\shape" ... Done.
Deleting the extracts "\tests\Leeds-shp" ... Done.
Deleting "tests\Leeds.osm.shp.zip" ... Done.

>>> osmdb_test.import_osm_data(leeds_shp, table_name=sr_name, verbose=True)
Confirmed to import the data into table "Leeds"
    at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Importing data into "Leeds" ...
    places ... done: <total of rows> features.
    waterways ... done: <total of rows> features.
    buildings ... done: <total of rows> features.
    natural ... done: <total of rows> features.
    roads ... done: <total of rows> features.
    railways ... done: <total of rows> features.
    points ... done: <total of rows> features.
    landuse ... done: <total of rows> features.
```

### 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 to be 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*, *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'
>>> dat_dir = "tests"

>>> # Import PBF data of Rutland

>>> rutland_pbf_raw = osmdb_test.Reader.read_osm_pbf(sr_name, dat_dir,
...                                                  verbose=True)
Confirmed to download .osm.pbf data of the following geographic region(s):
    Rutland
? [No]|Yes: yes

>>> tbl_name = sr_name
>>> schema = list(rutland_pbf_raw.keys())[0]  # 'points'

>>> rutland_pbf_raw_points = rutland_pbf_raw[schema]
>>> print(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...

>>> osmdb_test.import_osm_layer(rutland_pbf_raw_points, tbl_name, schema,
...                             verbose=True)
Confirmed to import the data into table '"points"."Rutland"'
    at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Creating a schema "points" ... Done.
Importing data into '"points"."Rutland"' ... Done.

>>> column_info = osmdb_test.get_subregion_table_column_info(tbl_name, schema)
>>> print(column_info.head())
                   column_0
table_catalog     osmdb_test
table_schema          points
table_name           Rutland
column_name           points
ordinal_position           1

>>> rutland_pbf = osmdb_test.Reader.read_osm_pbf(sr_name, dat_dir,
```

(continues on next page)

(continued from previous page)

```
...                                                 parse_raw_feat=True,
...                                                 transform_geom=True)

>>> rutland_pbf_points = rutland_pbf[schema]
>>> print(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]

>>> osmdb_test.import_osm_layer(rutland_pbf_points, tbl_name, schema,
...                             verbose=True)
Confirmed to import the data into table '"points"."Rutland"'
    at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
The table "points"."Rutland" already exists and is replaced ...
Importing data into '"points"."Rutland"' ... Done.

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

>>> # Import shapefile data of Rutland

>>> lyr_name = 'railways'
>>> rutland_railways_shp = osmdb_test.Reader.read_shp_zip(
...     sr_name, lyr_name, data_dir=dat_dir, rm_extracts=True,
...     rm_shp_zip=True, verbose=True)
Confirmed 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 from "rutland-latest-free.shp.zip" the following layer(s):
    'railways'
to "\tests\rutland-latest-free-shp" ...
In progress ... Done.
Deleting the extracts "\tests\rutland-latest-free-shp"  ... Done.
Deleting "tests\rutland-latest-free.shp.zip" ... Done.

>>> type(rutland_railways_shp)
<class 'dict'>
>>> print(list(rutland_railways_shp.keys()))
# ['railways']

>>> rutland_railways_shp_ = rutland_railways_shp[lyr_name]

>>> osmdb_test.import_osm_layer(rutland_railways_shp_, table_name=sr_name,
...                             schema_name=lyr_name, verbose=True)
Confirmed to import the data into table '"railways"."Rutland"'
    at postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Creating a schema "railways" ... Done.
Importing data into '"railways"."Rutland"' ... Done.

>>> col_info = osmdb_test.get_subregion_table_column_info(tbl_name, lyr_name)
>>> print(col_info.head())
```

(continues on next page)

(continued from previous page)

```
                    column_0    column_1  ...    column_6    column_7
table_catalog     osmdb_test  osmdb_test  ...  osmdb_test  osmdb_test
table_schema        railways    railways  ...    railways    railways
table_name           Rutland     Rutland  ...     Rutland     Rutland
column_name           osm_id        code  ...      tunnel    geometry
ordinal_position           1           2  ...           7           8
[5 rows x 8 columns]
```

### 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 .osm.pbf data file is located/saved; if `None`, 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 a `'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 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 `.import_osm_pbf_layer()`

**Examples**:

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

>>> dat_dir = "tests"

>>> sr_name = 'Rutland'

>>> osmdb_test.import_subregion_osm_pbf(sr_name, dat_dir, rm_osm_pbf=True,
...                                     verbose=True)
Confirmed to import Geofabrik OSM data of the following geographic region(s):
    Rutland
  into postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Downloading "rutland-latest.osm.pbf" to "\tests" ...
Done.
Importing data into "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.

>>> # Import free BBBike PBF data of Victoria and Waterloo
>>> 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)
Confirmed to import BBBike OSM data of the following geographic region(s):
    Victoria,
    Waterloo
  into postgres:***@localhost:5432/osmdb_test
? [No]|Yes: yes
Downloading "Victoria.osm.pbf" to "\tests" ...
Done.
```

(continues on next page)

(continued from previous page)

```
Parsing "\tests\Victoria.osm.pbf" ... Done.
Importing data into "Victoria" ...
    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 "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 data into "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.

>>> # The PBF data have also been saved as Pickle files
>>> victoria_pbf = load_pickle(cd(dat_dir, "Victoria-pbf.pickle"))
>>> print(victoria_pbf['points'].head())
         id                     coordinates  ... man_made other_tags
0  25832817  POINT (-123.3102145 48.4351935)  ...     None       None
1  25832953  POINT (-123.3157486 48.4309841)  ...     None       None
2  25832954  POINT (-123.3209612 48.4323984)  ...     None       None
3  25832995  POINT (-123.3224238 48.4321706)  ...     None       None
4  25833001  POINT (-123.3202181 48.4297225)  ...     None       None
[5 rows x 12 columns]

>>> waterloo_pbf = load_pickle(cd(dat_dir, "Waterloo-pbf.pickle"))
>>> print(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"))
```

### 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.

>>> sr_name = 'rutland'
>>> lyr_name = 'pt'

>>> # (If the table, pt."rutland", does not exist)
>>> osmdb_test.subregion_table_exists(sr_name, lyr_name)
False

# (If the table, points.'Rutland', does not exist)
>>> osmdb_test.subregion_table_exists(sr_name, lyr_name,
...                                   table_named_as_subregion=True,
...                                   schema_named_as_layer=True)
False
```

### Attributes

| | |
|---|---|
| *Downloader* | An instance of either *GeofabrikDownloader* or *BBBikeDownloader* depending on the specified `data_source` for creating a *PostgresOSM* instance. |
| *Name* | Name of the current *PostgresOSM.Downloader*. |
| *Reader* | An instance of either *GeofabrikReader* or *BBBikeReader* depending on the specified `data_source` for creating a *PostgresOSM* instance. |
| *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 a *PostgresOSM* instance.

#### PostgresOSM.Name

**property** PostgresOSM.**Name**
Name of the current *PostgresOSM.Downloader*.

#### PostgresOSM.Reader

**property** PostgresOSM.**Reader**
An instance of either *GeofabrikReader* or *BBBikeReader* depending on the specified `data_source` for creating a *PostgresOSM* instance.

#### PostgresOSM.URL

**property** PostgresOSM.**URL**
Homepage URL of data resource for current *PostgresOSM.Downloader*. :return:

### 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 = 'point'
>>> lyr_name_ = get_default_layer_name(lyr_name)

>>> 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** (*list 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

>>> schemas_names = validate_schema_names()

>>> print(schemas_names)
[]

>>> schemas_names_ = ['point', 'polygon']
>>> schemas_names = validate_schema_names(schemas_names_)

>>> print(schemas_names)
['point', 'polygon']

>>> schemas_names = validate_schema_names(schemas_names_,
...                                       schema_named_as_layer=True)

>>> print(schemas_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

>>> sr_name = 'greater london'
>>> tbl_name = validate_table_name(sr_name)

>>> print(tbl_name)
# greater london

>>> sr_name = 'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch, Wales'
>>> tbl_name = validate_table_name(sr_name, sub_space='_')

>>> print(tbl_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()
```

(continues on next page)

(continued from previous page)

```
>>> 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*) – 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

### 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_osm_geom_object_dict*() | A dictionary for OSM geometry 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_osm_geom_object_dict

pydriosm.utils.**get_osm_geom_object_dict**()

A dictionary for OSM geometry types.

**Returns** a dictionary with keys and values being shape types and shapely.geometry types

**Return type** dict

### 3.4.8 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_ | 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(). |

### 3.4.9 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']
```

### 3.4.10 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

### 3.4.11 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

### 3.4.12 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`

### 3.4.13 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

### 3.4.14 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()
```

## 3.6 updater

Updating package data.

| | |
|---|---|
| *update_package_data*([confirmation_requir ...]) | 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 to prompt a message 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 as the function runs, 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



## M



## N



## O



## P



## R



## S



## U



## V