
eGo^N Data
Release 0.0.0

Aug 17, 2023

Contents

1	About eGon-data	1
1.1	Project background	1
1.2	Objectives of the project	1
1.3	Project consortium and funding	1
1.4	eGon-data as one element of the eGo-Toolchain	2
1.5	Modeling concept and scenarios	2
2	Workflow	5
2.1	Workflow management	5
2.2	Execution	5
2.3	Versioning	5
3	Getting Started	7
3.1	Pre-requisites	7
3.2	Installation	8
3.3	Run the workflow	8
4	Troubleshooting	11
4.1	Installation Errors	11
4.2	Runtime Errors	11
4.3	Other import or incompatible package version errors	13
5	Data	15
5.1	Main input data and their processing	15
5.2	Grid models	17
5.3	Demand	20
5.4	Supply	27
5.5	Flexibility options	34
5.6	Published data	38
6	Literature	39
7	Contributing	41
7.1	Bug reports and feature requests	41
7.2	Contribution guidelines	42
7.3	Extending the data workflow	44
7.4	Documentation	46

8	Authors	49
9	Changelog	51
9.1	Unreleased	51
10	egon.data	63
10.1	airflow	63
10.2	cli	63
10.3	config	63
10.4	dataset_configuration	64
10.5	datasets	64
10.6	db	214
10.7	metadata	217
10.8	subprocess	219
11	Indices and tables	221
	Bibliography	223
	Python Module Index	225
	Index	229

1.1 Project background

egon-data provides a transparent and reproducible open data-based data processing pipeline for generating data models suitable for energy system modeling. The data is customized for the requirements of the research project eGo_n. The research project aims to develop tools for open and cross-sectoral planning of transmission and distribution grids. For further information please visit the [eGo_n project website](#). egon-data is a further development of the [Data processing](#) developed in the former research project [open_eGo](#). It aims to extend the data models as well as improve the replicability and manageability of the data preparation and processing. The resulting data set serves as an input for the optimization tools [eTraGo](#), [ding0](#) and [eDisGo](#) and delivers, for example, data on grid topologies, demands/demand curves and generation capacities in a high spatial resolution. The outputs of egon-data are published under open-source and open-data licenses.

1.2 Objectives of the project

Driven by the expansion of renewable generation capacity and the progressing electrification of other energy sectors, the electrical grid increasingly faces new challenges: fluctuating supply of renewable energy and simultaneously a changing demand pattern caused by sector coupling. However, the integration of non-electric sectors such as gas, heat, and e-mobility enables more flexibility options. The eGo_n project aims to investigate the effects of sector coupling on the electrical grid and the benefits of new flexibility options. This requires the creation of a spatially and temporally highly resolved database for all sectors considered.

1.3 Project consortium and funding

The following universities and research institutes were involved in the creation of eGon-data:

- University of Applied Sciences Flensburg
- Reiner Lemoine Institut

- Otto von Guericke University Magdeburg
- DLR Institute of Networked Energy Systems
- Europa-Universität Flensburg

The eGo_n project (FKZ: 03EI1002) is supported by the Federal Ministry for Economic Affairs and Climate Action (BMWK) on the basis of a decision by the German Bundestag.



1.4 eGon-data as one element of the eGo-Toolchain

In the eGo_n project different tools were developed, which are in exchange with each other and have to serve the respective requirements on data scope, resolution, and format. The results of the data model creation have to be especially adapted to the requirements of the tools eTraGo and eDisGo for power grid optimization on different grid levels. A PostgreSQL database serves as an interface between the data model creation and the optimization tools. The figure below visualizes the interdependencies between the different tools.

1.5 Modeling concept and scenarios

eGon-data provides a data model suitable for calculations and optimizations with the tools eTraGo, eDisGo and eGo and therefore aims to satisfy all requirements regarding the scope and temporal as well as spatial granularity of the resulting data model.

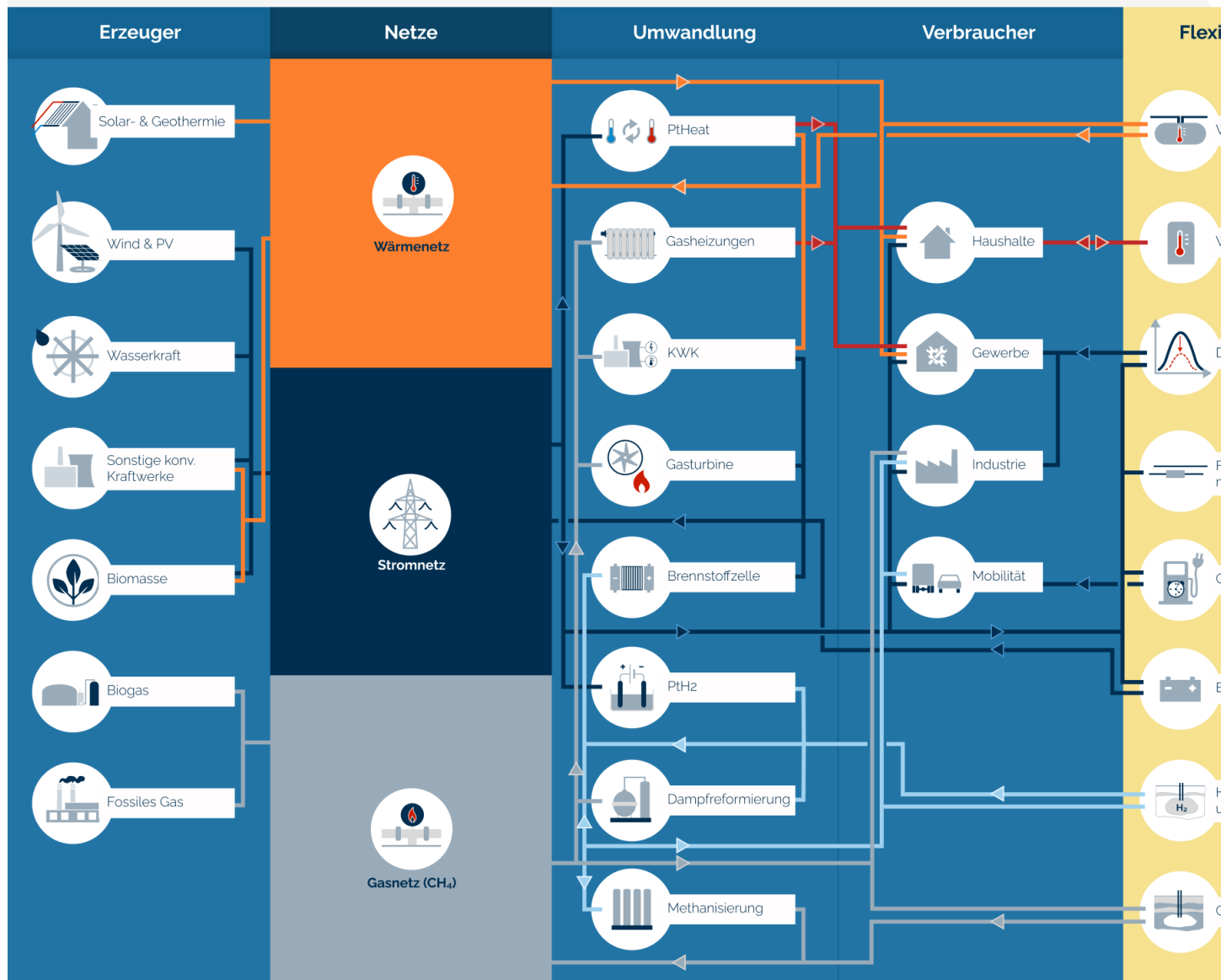
1.5.1 System boundaries and general assumptions

- Sectors
- Focus on Germany
- Neighbouring countries (which ones and why)
- Spatial resolution / aggregation levels
- Temporal resolution incl. assumptions on weather year

The following image visualizes the different components considered in scenario eGon2035.



Modell des Energiesystems: Szenario eGon2035



1.5.2 Scenarios

eGon-data aims to create different scenarios, which differ in terms of RE penetration or the availability of flexibility options. Currently, the following scenarios are available or in progress.

- eGon2035 Mid-termin scenario based on assumptions from the German network expansion plan ‘scenario C2035’, version 2021 and TYNDP
- eGon2035_lowflex Mid-termin scenario similar to ‘eGon2035’, but with a limited availability of flexibility options
- eGon100RE Long-term scenario with a 100% RE penetration, based on optimization results with PyPSA-Eur-Sec and additional data inputs (work-in-progress)

Table 1: Installed capacities of German power park in scenario eGon2035 and eGon2035_lowflex

carrier	Installed capacities
gas	46.7 GW
oil	1.3 GW
pumped hydro	10.2 GW
wind onshore	90.9 GW
wind offshore	34.0 GW
solar	120.1 GW
biomass	8.7 GW
others	5.4 GW

Table 2: German energy demands in scenarios eGon2035 and eGon2035_lowflex

Demand sector	Energy demand
MIT transport	41.4 TWh _{el}
central heat	68.9 TWh _{th}
rural heat	423.2 TWh _{th}
electricity	498.1 TWh _{el}
Methane industry	196.0 TWh _{CH4}
Hydrogen industry	16.1 TWh _{H2}
Hydrogen transport	26.5 TWh _{H2}

2.1 Workflow management

2.2 Execution

In principle egon-data is not limited to the use of a specific programming language as the workflow integrates different scripts using Apache Airflow, but Python and SQL are widely used within the process. Apache Airflow organizes the order of execution of processing steps through so-called operators. In the default case the SQL processing is executed on a containerized local PostgreSQL database using Docker. For further information on Docker and its installation please refer to their [documentation](#). Connection information of our local Docker database are defined in the corresponding [docker-compose.yml](#)

The egon-data workflow is composed of four different sections: database setup, data import, data processing and data export to the OpenEnergy Platform. Each section consists of different tasks, which are managed by Apache Airflow and correspond with the local database. Only final datasets which function as an input for the optimization tools or selected interim results are uploaded to the [Open Energy Platform](#). The data processing in egon-data needs to be performed locally as calculations on the Open Energy Platform are prohibited. More information on how to run the workflow can be found in the [getting started section](#) of our documentation.

2.3 Versioning

Warning: Please note, the following is not implemented yet, but we are working on it.

Source code and data are versioned independently from each other. Every data table uploaded to the Open Energy Platform contains a column ‘version’ which is used to identify different versions of the same data set. The version number is maintained for every table separately. This is a major difference to the versioning concept applied in the former data processing where all (interim) results were versioned under the same version number.

3.1 Pre-requisites

In addition to the installation of Python packages, some non-Python packages are required too. Right now these are:

- **Docker:** Docker is used to provide a PostgreSQL database (in the default case).
Docker provides extensive installation instruction. Best you consult [their docs](#) and choose the appropriate install method for your OS.
Docker is not required if you use a local PostgreSQL installation.
- The *psql* executable. On Ubuntu, this is provided by the *postgresql-client-common* package.
- Header files for the `libpq5` PostgreSQL library. These are necessary to build the `psycopg2` package from source and are provided by the `libpq-dev` package on Ubuntu.
- **osm2pgsql** On recent Ubuntu version you can install it via `sudo apt install osm2pgsql`.
- **postgis** On recent Ubuntu version you can install it via `sudo apt install postgis`.
- **osmTGmod** resp. **osmosis** needs **java**. On recent Ubuntu version you can install it via `sudo apt install default-jre` and `sudo apt install default-jdk`.
- **conda** is needed for the subprocess of running `pypsa-eur-sec`. For the installation of **miniconda**, check out the [conda installation guide](#).
- **pypsa-eur-sec** resp. **Fiona** needs the additional library `libtbb2`. On recent Ubuntu version you can install it via `sudo apt install libtbb2`
- **gdal** On recent Ubuntu version you can install it via `sudo apt install gdal-bin`.
- **curl** is required. You can install it via `sudo apt install curl`.
- To download ERA5 weather data you need to register at the CDS registration page and install the CDS API key as described [here](#) You also have to agree on the [terms of use](#)
- Make sure you have enough free disk space (~350 GB) in your working directory.

3.2 Installation

Since no release is available on PyPI and installations are probably used for development, cloning it via

```
git clone git@github.com:openego/eGon-data.git
```

and installing it in editable mode via

```
pip install -e eGon-data/
```

are recommended.

In order to keep the package installation isolated, we recommend installing the package in a dedicated virtual environment. There's both, an [external tool](#) and a [builtin module](#) which help in doing so. I also highly recommend spending the time to set up [virtualenvwrapper](#) to manage your virtual environments if you start having to keep multiple ones around.

If you run into any problems during the installation of `egon.data`, try looking into the list of [known installation problems](#) we have collected. Maybe we already know of your problem and also of a solution to it.

3.3 Run the workflow

The `egon.data` package installs a command line application called `egon-data` with which you can control the workflow so once the installation is successful, you can explore the command line interface starting with `egon-data --help`.

The most useful subcommand is probably `egon-data serve`. After running this command, you can open your browser and point it to `localhost:8080`, after which you will see the web interface of [Apache Airflow](#) with which you can control the *eGoⁿ* data processing pipeline.

If running `egon-data` results in an error, we also have collected a list of [known runtime errors](#), which can consult in search of a solution.

To run the workflow from the CLI without using `egon-data serve` you can use

```
egon-data airflow scheduler
egon-data airflow dags trigger egon-data-processing-pipeline
```

For further details how to use the CLI see [Apache Airflow CLI Reference](#).

Warning: A complete run of the workflow might require much computing power and can't be run on laptop. Use the *test mode* for experimenting.

Warning: A complete run of the workflow needs loads of free disk space (~350 GB) to store (temporary) files.

3.3.1 Test mode

The workflow can be tested on a smaller subset of data on example of the federal state of Schleswig-Holstein. Data is reduced during execution of the workflow to represent only this area.

Warning: Right now, the test mode is set in *egon.data/airflow/pipeline.py*.

Having trouble installing or running eGon-data? Here's a list of known issues including a solution.

4.1 Installation Errors

These are some errors you might encounter while trying to install *egon.data*.

4.1.1 `importlib_metadata.PackageNotFoundError: No package metadata ...`

It might happen that you have installed *importlib-metadata=3.1.0* for some reason which will lead to this error. Make sure you have *importlib-metadata>=3.1.1* installed. For more information read the discussion in [issue #60](#).

4.2 Runtime Errors

These are some of the errors you might encounter while trying to run *egon-data*.

4.2.1 `ERROR: Couldn't connect to Docker daemon ...`

To verify, please execute `docker-compose -f <(echo {"service": {"image": "hellow-world"}}) ps` and you should see something like

```
ERROR: Couldn't connect to Docker daemon at http+docker://localunixsocket - is it
↳running?
```

If it's at a non-standard location, specify the URL with the `DOCKER_HOST` environment variable.

This can have at least two possible reasons. First, the docker daemon might not be running. On Linux Systems, you can check for this by running `ps -e | grep dockerd`. If this generates no output, you have to start the docker daemon, which you can do via `sudo systemctl start docker.service` on recent Ubuntu systems.

Second, your current user might not be a member of the *docker* group. On Linux, you can check this by running `groups $(whoami)`. If the output does not contain the word *docker*, you have to add your current user to the *docker* group. You can find more information on how to do this in the [docker documentation](#). Read the [initial discussion](#) for more context.

4.2.2 [ERROR] Connection in use ...

This error might arise when running `egon-data serve` making it shut down early with `ERROR - Shutting down webserver`. The reason for this is that the local webserver from a previous `egon-data serve` run didn't shut down properly and is still running. This can be fixed by running `ps -eo pid,command | grep "unicorn: master" | grep -v grep` which should lead to output like `NUMBER unicorn: master [airflow-webserver]` where `NUMBER` is a varying number. Once you got this, run `kill -s INT NUMBER`, substituting `NUMBER` with what you got previously. After this, `egon-data serve` should run without errors again.

4.2.3 [ERROR] Cannot create container for service egon-data-local-database ...

During building the docker container for the Postgres database, you might encounter an error like

```
ERROR: for egon-data-local-database Cannot create container for service
egon-data-local-database: Conflict. The container name
"/egon-data-local-database" is already in use by container
"1ff9aade273a76a0acbf850c0da794d0fb28a30e9840f818ccala47d1181b00".
You have to remove (or rename) that container to be able to reuse that name.
```

If you're ok with deleting the data, stop and remove the container by

```
docker stop egon-data-local-database
docker rm -v egon-data-local-database
```

The container and its data can be kept by renaming the docker container.

```
docker rename egon-data-local-database NEW_CONTAINER_NAME
```

4.2.4 Working with multiple instances of egon-data

To make sure parallel installations of `egon-data` are not conflicting each other users have to set different values for the following options in the configuration:

```
--airflow-port
--compose-project-name
--database-port
--docker-container-name
```

4.3 Other import or incompatible package version errors

If you get an `ImportError` when trying to run `egon-data`, or the installation complains with something like

```
first-package a.b.c requires second-package>=q.r.r, but you'll have
second-package x.y.z which is incompatible.
```

you might have run into a problem of earlier `pip` versions. Either upgrade to a `pip` version `>=20.3` and reinstall `egon.data`, or reinstall the package via `pip install -U --use-feature=2020-resolver`. The `-U` flag is important to actually force a reinstall. For more information read the discussions in issues [#36](#) and [#37](#).

The description of the methods, input data and results of the eGon-data pipeline is given in the following section. References to datasets and functions are integrated if more detailed information is required.

5.1 Main input data and their processing

All methods in the eGon-data workflow rely on public and freely available data from different external sources. The most important data sources and their processing within the eGon-data pipeline are described here.

5.1.1 Data bundle

The data bundle is published on [zenodo](#). It contains several data sets, which serve as a basis for egon-data:

- Climate zones in Germany
- Data on eMobility individual trips of electric vehicles
- Spatial distribution of deep geothermal potentials in Germany
- Annual profiles in hourly resolution of electricity demand of private households
- Sample heat time series including hot water and space heating for single- and multi-family houses
- Hydrogen storage potentials in salt structures
- Information about industrial sites with DSM-potential in Germany
- Data extracted from the German grid development plan - power
- Parameters for the classification of gas pipelines
- Preliminary results from scenario generator pypsa-eur-sec
- German regions suitable to model dynamic line rating
- Eligible areas for wind turbines and ground-mounted PV systems

- Definitions of industrial and commercial branches
- Zensus data on households
- Geocoding of all unique combinations of ZIP code and municipality within the Marktstammdatenregister

For further description of the data including licenses and references please refer to the Zenodo repository.

5.1.2 Marktstammdatenregister

The [Marktstammdatenregister](#) (MaStR) is the register for the German electricity and gas market holding, among others, data on electricity and gas generation plants. In eGon-data it is used for status quo data on PV plants, wind turbines, biomass, hydro power plants, combustion power plants, nuclear power plants, geo- and solarthermal power plants, and storage units. The data are obtained from zenodo, where raw MaStR data, downloaded with the tool [open-MaStR](#) using the MaStR webservice, is provided. It contains all data from the MaStR, including possible duplicates. Currently, two versions are used:

- 2021-05-03
- 2022-11-17

The download is implemented in `MastrData`.

5.1.3 OpenStreetMap

[OpenStreetMap](#) (OSM) is a free, editable map of the whole world that is being built by volunteers and released with an open-content license. In eGon-data it is, among others, used to obtain information on land use as well as locations of buildings and amenities to spatially dissolve energy demand. The OSM data is downloaded from the [Geofabrik](#) download server, which holds extracts from the OpenStreetMap. Afterwards, they are imported to the database using `osm2pgsql` with a custom style file. The implementation of this can be found in [OpenStreetMap](#).

In the [OpenStreetMap](#) dataset, the OSM data is filtered, processed and enriched with other data. This is described in the following subsections.

Amenity data

The data on amenities is used to disaggregate CTS demand data. It is filtered from the raw OSM data using tags listed in script `osm_amenities_shops_preprocessing.sql`, e.g. shops and restaurants. The filtered data is written to database table `openstreetmap.osm_amenities_shops_filtered`.

Building data

The data on buildings is required by several tasks in the pipeline, such as the disaggregation of household demand profiles or PV home systems to buildings, as well as the DIstribution Network Generator [ding0](#) (see also `ding0-grids`).

The data processing steps are:

- Extract buildings and filter using relevant tags, e.g. residential and commercial, see script `osm_buildings_filter.sql` for the full list of tags. Resulting tables:
 - All buildings: `openstreetmap.osm_buildings`
 - Filtered buildings: `openstreetmap.osm_buildings_filtered`
 - Residential buildings: `openstreetmap.osm_buildings_residential`

- Create a mapping table for building's OSM IDs to the Zensus cells the building's centroid is located in. Resulting tables:
 - *boundaries.egon_map_zensus_buildings_filtered* (filtered)
 - *boundaries.egon_map_zensus_buildings_residential* (residential only)
- Enrich each building by number of apartments from Zensus table *society.egon_destatis_zensus_apartment_building_population_per_ha* by splitting up the cell's sum equally to the buildings. In some cases, a Zensus cell does not contain buildings but there is a building nearby which the no. of apartments is to be allocated to. To make sure apartments are allocated to at least one building, a radius of 77m is used to catch building geometries.
- Split filtered buildings into 3 datasets using the amenities' locations: temporary tables are created in script *osm_buildings_temp_tables.sql*, the final tables in *osm_buildings_amenities_results.sql*. Resulting tables:
 - Buildings w/ amenities: *openstreetmap.osm_buildings_with_amenities*
 - Buildings w/o amenities: *openstreetmap.osm_buildings_without_amenities*
 - Amenities not allocated to buildings: *openstreetmap.osm_amenities_not_in_buildings*

As there are discrepancies between the Census data [Census] and OSM building data when both datasets are used to generate electricity demand profiles of households, synthetic buildings are added in Census cells with households but without buildings. This is done as part of the *Demand_Building_Assignment* dataset in function *generate_synthetic_buildings*. The synthetic building data are written to table *openstreetmap.osm_buildings_synthetic*. The same is done in case of CTS electricity demand profiles. Here, electricity demand is disaggregated to Census cells according to heat demand information from the Pan European Thermal Atlas [Peta]. In case there are Census cells with electricity demand assigned but no building or amenity data, synthetic buildings are added. This is done as part of the *CtsDemandBuildings* dataset in function *create_synthetic_buildings*. The synthetic building data are again written to table *openstreetmap.osm_buildings_synthetic*.

Street data

The data on streets is used in the DIstribution Network Generator *ding0*, e.g. for the routing of the grid. It is filtered from the raw OSM data using tags listed in script *osm_ways_preprocessing.sql*, e.g. *highway=secondary*. Additionally, each way is split into its line segments and their lengths is retained. The filtered streets data is written to database table *openstreetmap.osm_ways_preprocessed* and the filtered streets with segments to table *openstreetmap.osm_ways_with_segments*.

5.2 Grid models

Power grid models of different voltage levels form a central part of the eGon data model, which is required for cross-grid-level optimization. In addition, sector coupling necessitates the representation of the gas grid infrastructure, which is also described in this section.

5.2.1 Electricity grid

High and extra-high voltage grids

The model of the German extra-high (eHV) and high voltage (HV) grid is based on data retrieved from OpenStreetMap (OSM) (status January 2021) [OSM] and additional parameters for standard transmission lines from [Brakelmann2004]. To gather all required information, such as line topology, voltage level, substation locations, and

electrical parameters, to create a calculable power system model, the **osmTGmod** tool was used. The corresponding dataset *osmtgmod* executes osmTGmod and writes the resulting data to the database.

The resulting grid model includes the voltage levels 380, 220 and 110 kV and all substations interconnecting the different grid levels. Information about border crossing lines are as well extracted from OSM data by *osmTGmod*. For further information on the generation of the grid topology please refer to [Mueller2018]. The neighbouring countries are included in the model in a significantly lower spatial resolution with one or two nodes per country. The border crossing lines extracted by *osmTGmod* are extended to representative nodes of the respective country in dataset *ElectricalNeighbours*. The resulting grid topology is shown in the following figure.

..figure:: images/Stromnetz.png

scale 50 %

name gridtopology_ehv_hv

alt Grid topology extra-high and high voltage grid

Medium and low-voltage grids

Medium (MV) and low (LV) voltage grid topologies for entire Germany are generated using the python tool *ding0*. *ding0* generates synthetic grid topologies based on high-resolution geodata and routing algorithms as well as typical network planning principles. The generation of the grid topologies is not part of *eGon_data*, but *ding0* solely uses data generated with *eGon_data*, such as locations of HV/MV stations (see *ehv-hv-grids*), locations and peak demands of buildings in the grid (see *building-data-ref* respectively *Electricity*), as well as locations of generators from MaStR (see *mastr-ref*). A full list of tables used in *ding0* can be found in its *config*. An exemplary MV grid with one underlying LV grid is shown in figure *ding0-mv-grid*. The grid data of all over 3.800 MV grids is published on *zenodo*.

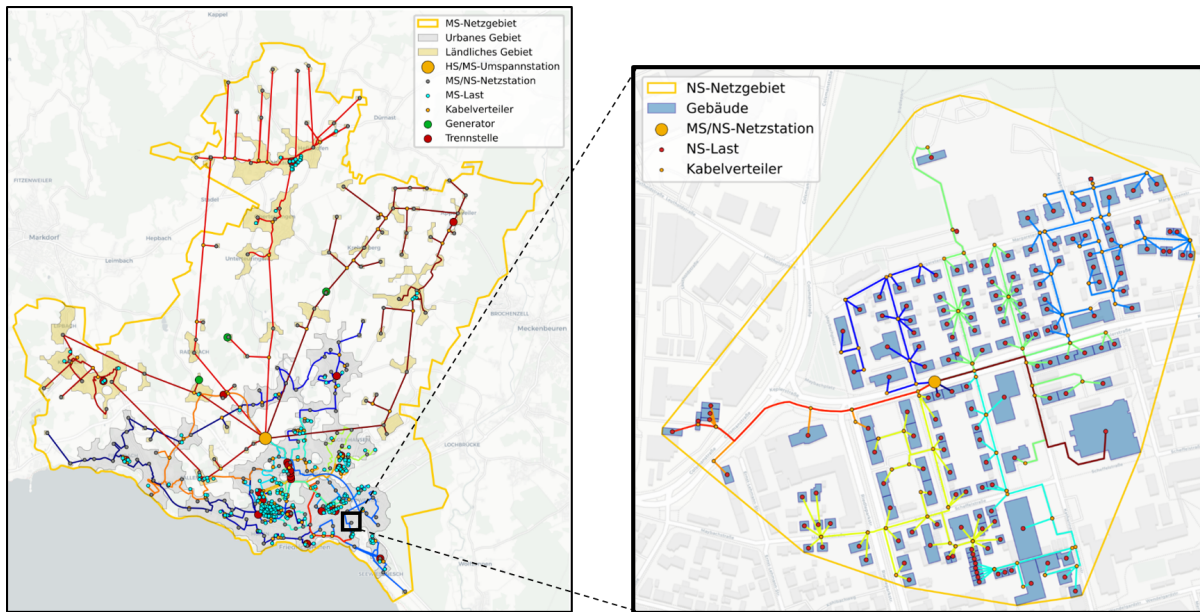


Fig. 1: Exemplary synthetic medium-voltage grid with underlying low-voltage grid generated with *ding0*

Besides data on buildings and generators, *ding0* requires data on the supplied areas by each grid. This is as well done in *eGon_data* and described in the following.

MV grid districts

Medium-voltage (MV) grid districts describe the area supplied by one MV grid. They are defined by one polygon that represents the supply area. Each MV grid district is connected to the HV grid via a single substation. An exemplary MV grid district is shown in figure ding0-mv-grid (orange line).

The MV grid districts are generated in the dataset *MvGridDistricts*. The methods used for identifying the MV grid districts are heavily inspired by Hülk et al. (2017) [Huelk2017] (section 2.3), but the implementation differs in detail. The main difference is that direct adjacency is preferred over proximity. For polygons of municipalities without a substation inside, it is iteratively checked for direct adjacent other polygons that have a substation inside. Speaking visually, a MV grid district grows around a polygon with a substation inside.

The grid districts are identified using three data sources

1. Polygons of municipalities (*Vg250GemClean*)
2. Locations of HV-MV substations (*EgonHvmvSubstation*)
3. HV-MV substation voronoi polygons (*EgonHvmvSubstationVoronoi*)

Fundamentally, it is assumed that grid districts (supply areas) often go along borders of administrative units, in particular along the borders of municipalities due to the concession levy. Furthermore, it is assumed that one grid district is supplied via a single substation and that locations of substations and grid districts are designed for aiming least lengths of grid line and cables.

With these assumptions, the three data sources from above are processed as follows:

- Find the number of substations inside each municipality
- Split municipalities with more than one substation inside
 - Cut polygons of municipalities with voronoi polygons of respective substations
 - Assign resulting municipality polygon fragments to nearest substation
- Assign municipalities without a single substation to nearest substation in the neighborhood
- Merge all municipality polygons and parts of municipality polygons to a single polygon grouped by the assigned substation

For finding the nearest substation, as already said, direct adjacency is preferred over closest distance. This means, the nearest substation does not necessarily have to be the closest substation in the sense of beeline distance. But it is the substation definitely located in a neighboring polygon. This prevents the algorithm to find solutions where a MV grid districts consists of multi-polygons with some space in between. Nevertheless, beeline distance still plays an important role, as the algorithm acts in two steps

1. Iteratively look for neighboring polygons until there are no further polygons
2. Find a polygon to assign to by minimum beeline distance

The second step is required in order to cover edge cases, such as islands.

For understanding how this is implemented into separate functions, please see *define_mv_grid_districts*.

Load areas

Load areas (LAs) are defined as geographic clusters where electricity is consumed. They are used in ding0 to determine the extent and number of LV grids. Thus, within each LA there are one or multiple MV-LV substations, each supplying one LV grid. Exemplary load areas are shown in figure ding0-mv-grid (grey and orange areas).

The load areas are set up in the *LoadArea* dataset. The methods used for identifying the load areas are heavily inspired by Hülk et al. (2017) [Huelk2017] (section 2.4).

5.2.2 Gas grid

Information about the gas grids and how they were created

Methane grid

Hydrogen grid

5.3 Demand

Electricity, heat and gas demands from different consumption sectors are taken into account in eGon-data. The related methods to distribute and process the demand data are described in the following chapters for the different consumption sectors separately.

5.3.1 Electricity

The electricity demand considered includes demand from the residential, commercial and industrial sector. The target values for scenario *eGon2035* are taken from the German grid development plan from 2021 [NEP2021], whereas the distribution on NUTS3-levels corresponds to the data from the research project *DemandRegio* [demandregio]. The following table lists the electricity demands per sector:

Table 1: Electricity demand per sector

Sector	Annual electricity demand in TWh
residential	115.1
commercial	123.5
industrial	259.5

A further spatial and temporal distribution of the electricity demand is needed to fulfil all requirements of the subsequent grid optimization. Therefore different, sector-specific distributions methods were developed and applied.

The annual electricity demands of households on NUTS3-level from *DemandRegio* are scaled to meet the national target values for the respective scenario in dataset *DemandRegio*. A further spatial and temporal distribution of residential electricity demands is performed in *HouseholdElectricityDemand* as described in [Buettner2022]. The result is a consistent dataset across aggregation levels with an hourly resolution.

The distribution of electricity demand from the commercial, trade and service (CTS) sector is also based on data from *DemandRegio*, which provides annual electricity demands on NUTS3-level for Germany. In dataset *CtsElectricityDemand* the annual electricity demands are further distributed to census cells (100x100m cells from [Census]) based on the distribution of heat demands, which is taken from the Pan-European Thermal Atlas version 5.0.1 [Peta]. For further information refer to section ref:heat_demand. The applied methods for a further spatial and temporal distribution to buildings is described in [Buettner2022] and performed in dataset *CtsDemandBuildings*

To distribute the annual industrial electricity demand OSM landuse data as well as information on industrial sites are taken into account. In a first step (*CtsElectricityDemand*) different sources providing information about specific sites and further information on the industry sector in which the respective industrial site operates are combined. Here, the three data sources [Hotmaps], [sEEnergies] and [Schmidt2018] are aligned and joined. Based on the resulting list of industrial sites in Germany and information on industrial landuse areas from OSM [OSM] which were extracted and processed in *OsmLanduse* the annual demands were distributed. The spatial and temporal distribution is performed in *IndustrialDemandCurves*. For the spatial distribution of annual electricity demands from *DemandRegio* [demandregio] which are available on NUTS3-level are in a first step evenly split 50/50 between industrial



Fig. 2: Electricity demand on NUTS 3-level (upper left); Exemplary MVGD (upper right); Study region in Flensburg (20 Census cells, bottom) from [Buettner2022]



Fig. 3: Electricity demand time series on different aggregation levels from [Buettner2022]

sites and OSM-polygons tagged as industrial areas. Per NUTS-3 area the respective shares are then distributed linearly based on the area of the corresponding landuse polygons and evenly to the identified industrial sites. In a next step the temporal disaggregation of the annual demands is carried out taking information about the industrial sectors and sector-specific standard load profiles from [demandregio] into account. Based on the resulting time series and their peak loads the corresponding grid level and grid connections point is identified.

The neighbouring countries considered in the model are represented in a lower spatial resolution of one or two buses per country. The national demand timeseries in an hourly resolution of the respective countries is taken from the Ten-Year Network Development Plan, Version 2020 [TYNDP]. In case no data for the target year is available the data is interpolated linearly. Refer to the corresponding dataset for detailed information: `ElectricalNeighbours`

5.3.2 Heat

Heat demands comprise space heating and drinking hot water demands from residential and commercial trade and service (CTS) buildings. Process heat demands from the industry are, depending on the required temperature level, modelled as electricity, hydrogen or methane demand.

The spatial distribution of annual heat demands is taken from the Pan-European Thermal Atlas version 5.0.1 [Peta]. This source provides data on annual European residential and CTS heat demands per hectare cell for the year 2015. In order to model future demands, the demand distribution extracted by Peta is then scaled to meet a national annual demand from external sources. The following national demands are taken for the selected scenarios:

Table 2: Heat demands per sector and scenario

	Residential sector	CTS sector	Sources
eGon2035	379 TWh	122 TWh	[Energierferenzprognose]
eGon100RE	284 TWh	89 TWh	[Energierferenzprognose]

The resulting data is stored in the database table `demand.egon_peta_heat`. The implementation of these data-processing steps can be found in `HeatDemandImport`.

Figure residential-heat-demand-annual shows the distribution of residential heat demands for scenario eGon2035, categorized for different levels of annual demands.

Afterwards, the annual heat demands are used to create hourly heat demand profiles. For residential heat demand profiles a pool of synthetically created bottom-up demand profiles is used. Depending on the mean temperature per day, these profiles are randomly assigned to each residential building. The methodology is described in detail in [Buettner2022].

Data on residential heat demand profiles is stored in the database within the tables `demand.egon_heat_timeseries_selected_profiles`, `demand.egon_daily_heat_demand_per_climate_zone`, `boundaries.egon_map_zensus_climate_zones`. To create the profiles for a selected building, these tables have to be combined, e.g. like this:

```
SELECT (b.demand/f.count * UNNEST(e.idp) * d.daily_demand_share)*1000 AS demand_
↪profile
FROM (SELECT * FROM demand.egon_heat_timeseries_selected_profiles,
UNNEST(selected_idp_profiles) WITH ORDINALITY as selected_idp) a
JOIN demand.egon_peta_heat b
ON b.zensus_population_id = a.zensus_population_id
JOIN boundaries.egon_map_zensus_climate_zones c
ON c.zensus_population_id = a.zensus_population_id
JOIN demand.egon_daily_heat_demand_per_climate_zone d
ON (c.climate_zone = d.climate_zone AND d.day_of_year = ordinality)
JOIN demand.egon_heat_idp_pool e
ON selected_idp = e.index
```

(continues on next page)

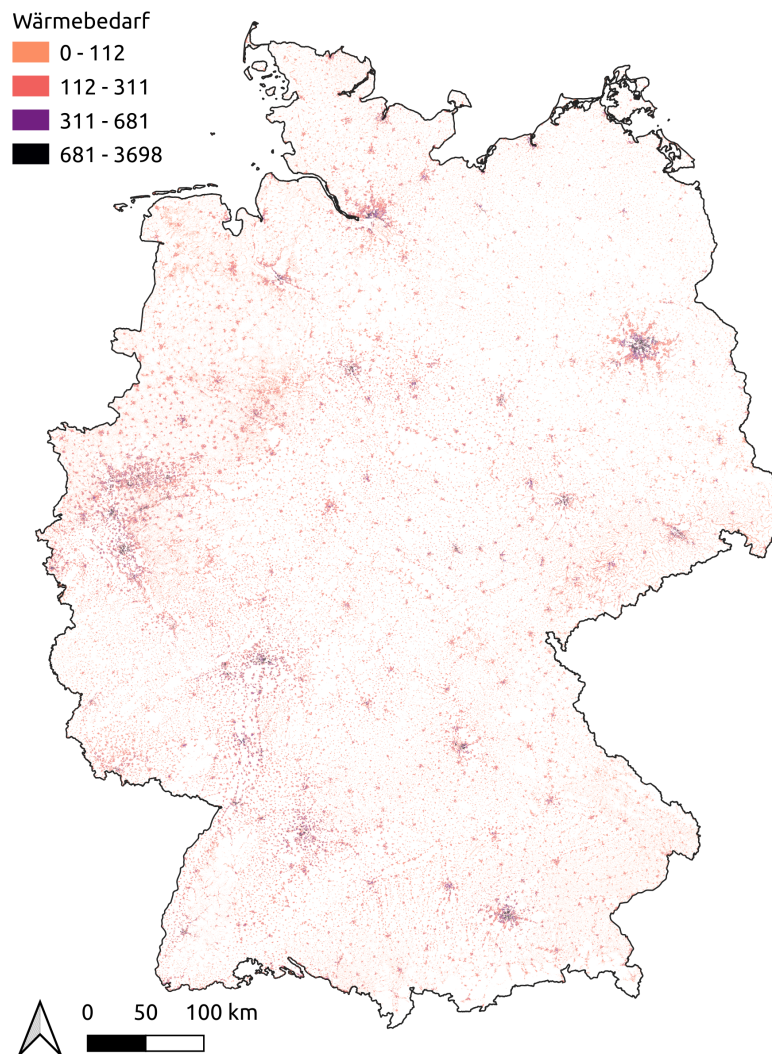


Fig. 4: Spatial distribution of residential heat demand in scenario eGon2035

(continued from previous page)

```

JOIN (SELECT zensus_population_id, COUNT(building_id)
FROM demand.egon_heat_timeseries_selected_profiles
GROUP BY zensus_population_id
) f
ON f.zensus_population_id = a.zensus_population_id
WHERE a.building_id = SELECTED_BUILDING_ID
AND b.scenario = 'eGon2035'
AND b.sector = 'residential';

```

Exemplary resulting residential heat demand time series for a selected day in winter and summer considering different aggregation levels are visualized in figures residential-heat-demand-timeseries-winter and residential-heat-demand-timeseries-summer.

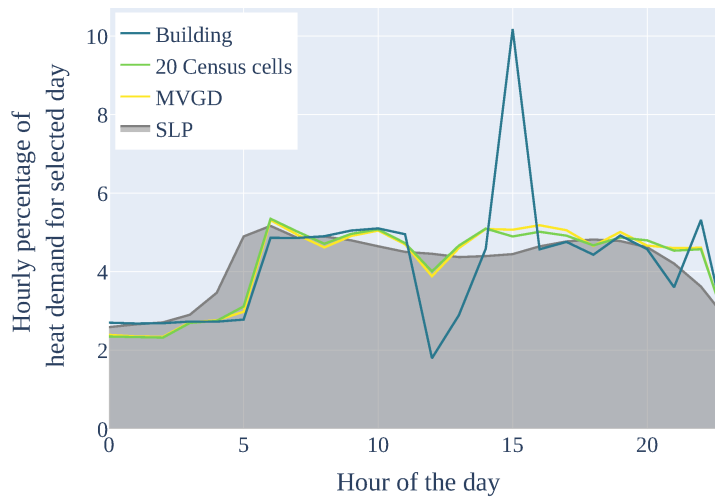


Fig. 5: Temporal distribution of residential heat demand for a selected day in winter

The temporal disaggregation of CTS heat demand is done using Standard Load Profiles Gas from `demandregio` [`demandregio`] considering different profiles per CTS branch.

The heat demand time series for both sectors creation is done in the Dataset *HeatTimeSeries*.

5.3.3 Gas

Information about gas demands and their spatial and temporal aggregation, including hydrogen and methane demands

5.3.4 Mobility

Motorized individual travel

The electricity demand data of motorized individual travel (MIT) for both the eGon2035 and eGon100RE scenario is set up in the `MotorizedIndividualTravel` dataset.

The profiles are generated using a modified version of `SimBEV v0.1.3`. SimBEV generates driving profiles for battery electric vehicles (BEVs) and plug-in hybrid electric vehicles (PHEVs) based on MID survey data [`MiD2017`] per

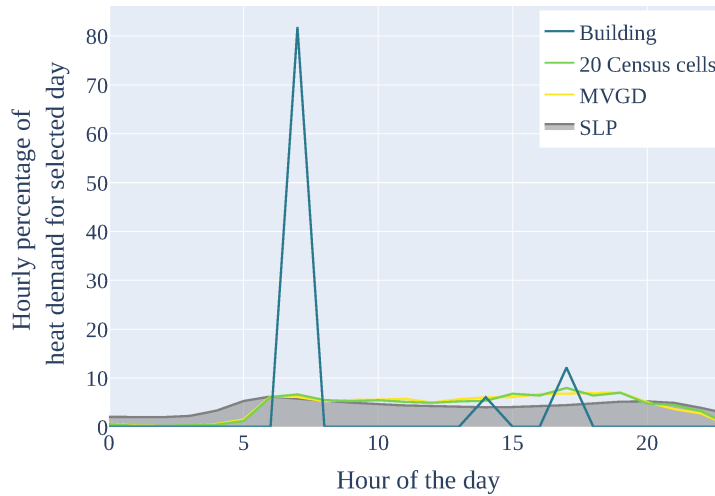


Fig. 6: Temporal distribution of residential heat demand for a selected day in summer

RegioStaR7 region type [RegioStaR7_2020]. These profiles include driving, parking and (user-oriented) charging times. Different vehicle classes are taken into account whose assumed technical data is given in table `ev-types-data`. Moreover, charging probabilities for multiple types of charging infrastructure are presumed based on [NOW2020] and [Helfenbein2021]. Given these assumptions, a pool of 33.000 EVs-types is pre-generated and provided through the data bundle (see `data-bundle-ref`) as well as written to table `EgonEvTrip`. The complete tech data and assumptions of the run can be found in the `metadata_simbev_run.json` file, that is provided along with the trip data.

Table 3: EV types

Technology	Size	Max. charging capacity slow in kW	Max. charging capacity fast in kW	Battery capacity in kWh	Energy consumption in kWh/km
BEV	mini	11	120	60	0.1397
BEV	medium	22	350	90	0.1746
BEV	luxury	50	350	110	0.2096
PHEV	mini	3.7	40	14	0.1425
PHEV	medium	11	40	20	0.1782
PHEV	luxury	11	120	30	0.2138

Heavy-duty transport

In the context of the eGon project, it is assumed that all e-trucks will be completely hydrogen-powered. The hydrogen demand data of all e-trucks is set up in the `HeavyDutyTransport` dataset for both the eGon2035 and eGon100RE scenario.

In both scenarios the hydrogen consumption is assumed to be 6.68 kgH₂ per 100 km with an additional supply chain leakage rate of 0.5 % (see [here](#)).

For the eGon2035 scenario the ramp-up figures are taken from the [network development plan \(version 2021\)](#) (Scenario C 2035). According to this, 100,000 e-trucks are expected in Germany in 2035, each covering an average of 100,000

km per year. In total this means 10 Billion km.

For the eGon100RE scenario it is assumed that the heavy-duty transport is completely hydrogen-powered. The total freight traffic with 40 Billion km is taken from the [BMWK Langfristszenarien](#) for heavy-duty vehicles larger 12 t allowed total weight (SNF > 12 t zGG).

The total hydrogen demand is spatially distributed on the basis of traffic volume data from [BAsT]. For this purpose, first a voronoi partition of Germany using the traffic measuring points is created. Afterwards, the spatial shares of the Voronoi regions in each NUTS3 area are used to allocate hydrogen demand to the NUTS3 regions and are then aggregated per NUTS3 region. The refuelling is assumed to take place at a constant rate. Finally, to determine the hydrogen bus where the hydrogen demand is allocated to, the centroid of each NUTS3 region is used to determine the respective hydrogen Voronoi cell (see [GasAreaseGon2035](#) and [GasAreaseGon100RE](#)) it is located in.

5.4 Supply

The distribution and assignment of supply capacities or potentials are carried out technology-specific. The different methods are described in the following chapters.

5.4.1 Electricity

The electrical power plants park, including data on geolocations, installed capacities, etc. for the different scenarios is set up in the dataset [PowerPlants](#).

Main inputs into the dataset are target capacities per technology and federal state in each scenario (see [Modeling concept and scenarios](#)) as well as the MaStR (see [mastr-ref](#)), OpenStreetMap (see [osm-ref](#)) and potential areas (provided through the data bundle, see [data-bundle-ref](#)) to distribute the generator capacities within each federal state region. The approach taken to distribute the target capacities within each federal state differs for the different technologies and is described in the following. The final distribution in the eGon2035 scenario is shown in figure [generator-park-egon-2035](#).

Onshore wind

Offshore wind

PV ground mounted

The distribution of PV ground mounted is implemented in function [insert](#) which is part of the dataset [PowerPlants](#). The following steps are conducted:

1. The sites and capacities of existing PV parks are imported using MaStR data (see [mastr-ref](#)).
2. Potential areas for PV ground mounted are assumed to be areas next to highways and railways as well as on agricultural land with a low degree of utilisation, as it can be seen in figure [pv_ground_mounted-example](#). Those areas (provided through the data bundle, see [data-bundle-ref](#)) are imported while merging or discarding small areas.
3. The locations of existing parks and the potential areas are intersected with each other while considering a buffer around the locations of existing parks to find out where there already are parks at or close to potential areas. This results in a selection of potential areas.
4. The capacities of the existing parks are considered and compared to the target values for the specific scenario per federal state (see [Modeling concept and scenarios](#)). The required expansion capacity is derived.
5. If expansion of PV ground mounted capacity is required, capacities are calculated depending on the area size of the formerly selected potential areas. The resulting parks are therefore located on the selected potential areas.

Kraftwerkspark

- solar rooftop
- solar ground
- wind_onshore
- wind_offshore
- biomass
- gas
- oil
- reservoir
- run_of_river

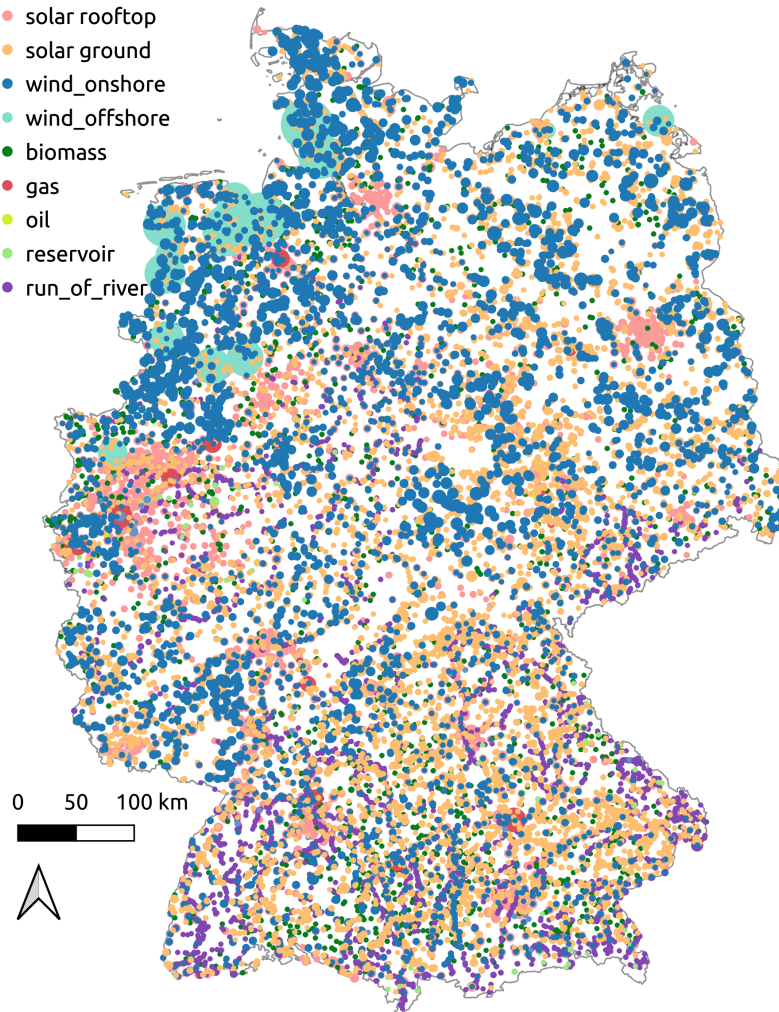


Fig. 7: Generator park in the eGon2035 scenario

6. The resulting capacities are compared to the target values for the specific scenario per federal state. If the target value is exceeded, a linear downscaling is conducted. If the target value is not reached yet, the remaining capacity is distributed linearly among the rest of the potential areas within the state.

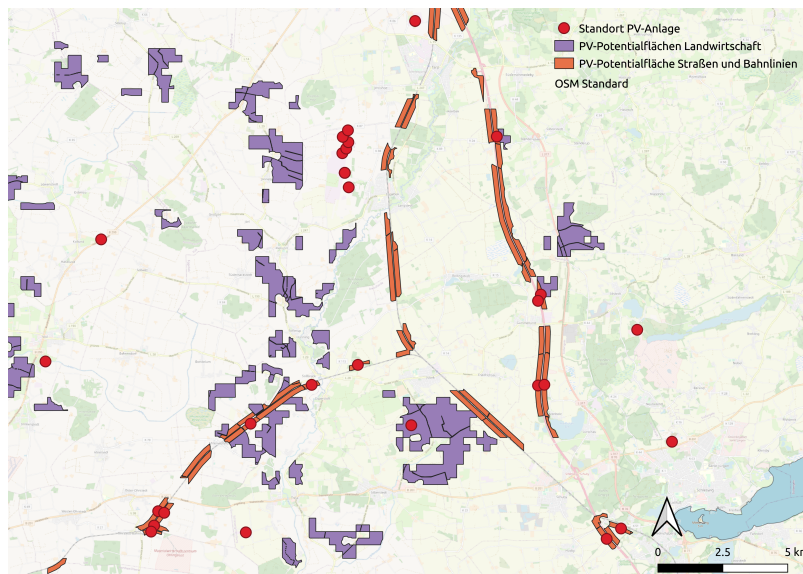


Fig. 8: Example: sites of existing PV ground mounted parks and potential areas

PV rooftop

In a first step, the target capacity in the eGon2035 and eGon100RE scenarios is distributed to all MV grid districts linear to the residential and CTS electricity demands in the grid district (done in function `pv_rooftop_per_mv_grid`).

Afterwards, the PV rooftop capacity per MV grid district is disaggregated to individual buildings inside the grid district (done in function `pv_rooftop_to_buildings`). The basis for this is data from the MaStR, which is first cleaned and missing information inferred, and then allocated to specific buildings. New PV plants are in a last step added based on the capacity distribution from MaStR. These steps are in more detail described in the following.

MaStR data cleaning and inference:

- Drop duplicates and entries with missing critical data.
- Determine most plausible capacity from multiple values given in MaStR data.
- Drop generators that don't have a plausible capacity ($23.5 \text{ MW} > P > 0.1 \text{ kW}$).
- Randomly and weighted add a start-up date if it is missing.
- Extract zip and municipality from 'site' given in MaStR data.
- Geocode unique zip and municipality combinations with Nominatim (1 sec delay). Drop generators for which geocoding failed or which are located outside the municipalities of Germany.
- Add some visual sanity checks for cleaned data.

Allocation of MaStR plants to buildings:

- Allocate each generator to an existing building from OSM or a synthetic building (see building-data-ref).
- Determine the quantile each generator and building is in depending on the capacity of the generator and the area of the polygon of the building.

- Randomly distribute generators within each municipality preferably within the same building area quantile as the generators are capacity wise.
- If not enough buildings exist within a municipality and quantile additional buildings from other quantiles are chosen randomly.

Disaggregation of PV rooftop scenario capacities:

- The scenario data per federal state is linearly distributed to the MV grid districts according to the PV rooftop potential per MV grid district.
- The rooftop potential is estimated from the building area given from the OSM buildings.
- Grid districts, which are located in several federal states, are allocated PV capacity according to their respective roof potential in the individual federal states.
- The disaggregation of PV plants within a grid district respects existing plants from MaStR, which did not reach their end of life.
- New PV plants are randomly and weighted generated using the capacity distribution of PV rooftop plants from MaStR.
- Plant metadata (e.g. plant orientation) is also added randomly and weighted using MaStR data as basis.

Hydro

In the case of hydropower plants, a distinction is made between the carrier run-of-river and reservoir. The methods to distribute and allocate are the same for both carriers. In a first step all suitable power plants (correct carrier, valid geolocation, information about federal state) are selected and their installed capacity is scaled to meet the target values for the respective federal state and scenario. Information about the voltage level the power plants are connected to is obtained. In case no information is available the voltage level is identified using threshold values for the installed capacity (see `assign_voltage_level`). In a next step the correct grid connection point is identified based on the voltage level and geolocation of the power plants (see `assign_bus_id`). The resulting list of power plants is added to table `EgonPowerPlants`.

Biomass

The allocation of biomass-based power plants follows the same method as the one for hydro power plants and is performed in function `insert_biomass_plants`

Conventional

CHP

non-chp

In function `allocate_conventional_non_chp_power_plants` capacities for conventional power plants, which are no chp plants, with carrier *oil* and *gas* are allocated.

5.4.2 Heat

Heat demand of residential as well as commercial, trade and service (CTS) buildings can be supplied by different technologies and carriers. Within the data model creation, capacities of supply technologies are assigned to specific locations and their demands. The hourly dispatch of heat supply is not part of the data model, but a result of the grid optimization tools.

In general, heat supply can be divided into three categories which include specific technologies: residential and CTS buildings in a district heating area, buildings supplied by individual heat pumps, and buildings supplied by conventional gas boilers. The shares of these categories are taken from external sources for each scenario.

Table 4: Heat demands of different supply categories

	District heating	Individual heat pumps	Individual gas boilers
eGon2035	69 TWh	27.24 TWh	390.78 TWh
eGon100RE	61.5 TWh	311.5 TWh	0 TWh

The following subsections describe the heat supply methodology for each category.

First, district heating areas are defined for each scenario based on existing district heating areas and an overall district heating share per scenario. To reduce the model complexity, district heating areas are defined per Census cell, either all buildings within a cell are supplied by district heat or none. The first step of the extraction of district heating areas is the identification of Census cells with buildings that are currently supplied by district heating using the *building* dataset of Census. All Census cells where more than 30% of the buildings are currently supplied by district heat are defined as cells inside a district heating area. The identified cells are then summarized by combining cells that have a maximum distance of 500m.

Second, additional Census cells are assigned to district heating areas considering the heat demand density. Assuming that new district heating grids are more likely in cells with high demand, the remaining Census cells outside of a district heating grid are sorted by their demands. Until the pre-defined national district heating demand is met, cells from that list are assigned to district heating areas. This can also result in new district heating grids which cover only a few Census cells.

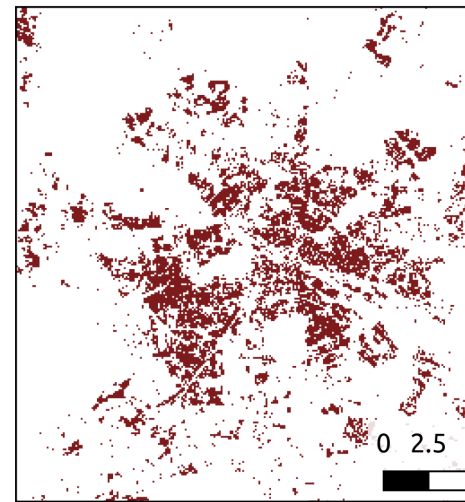
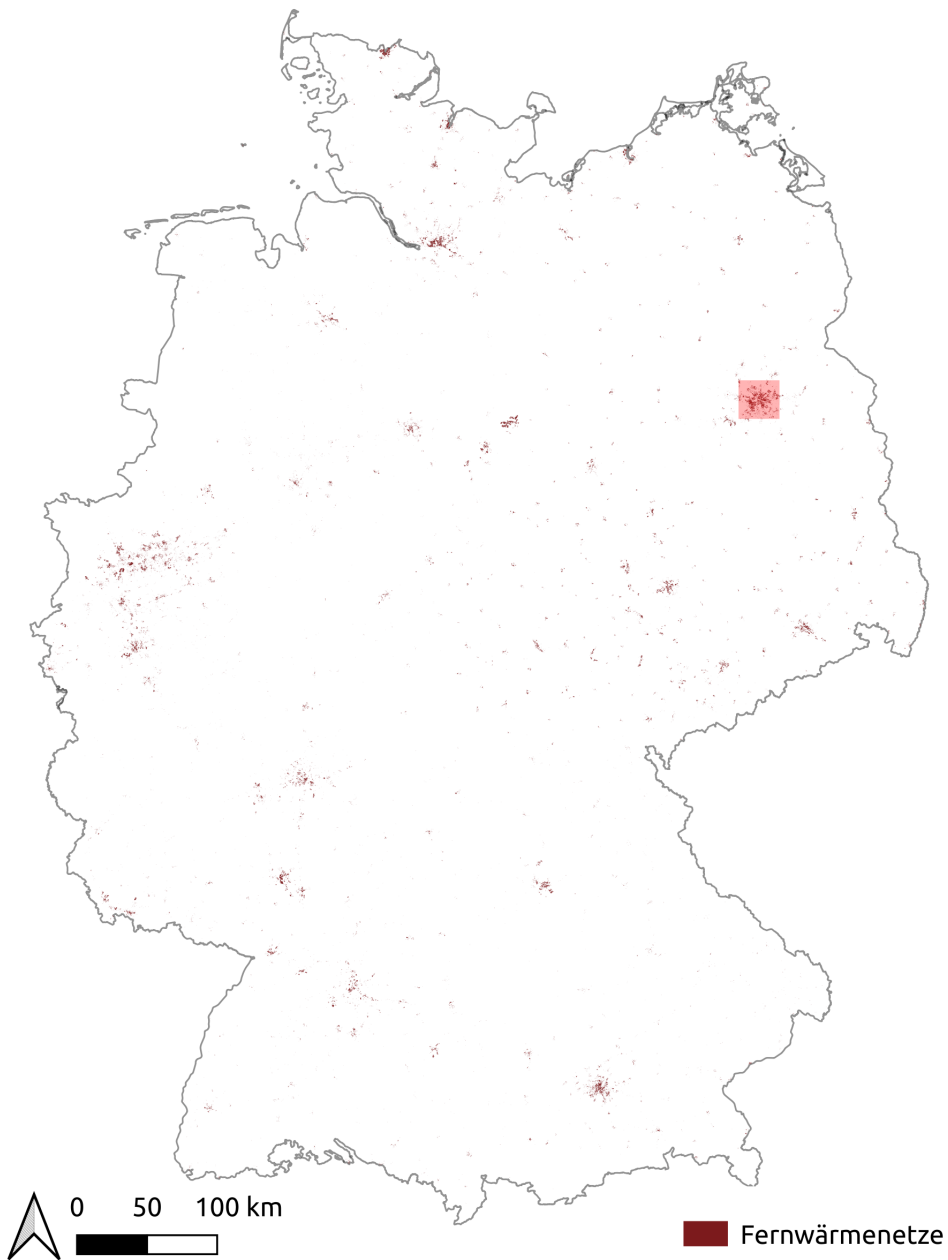
To avoid unrealistic large district heating grids in areas with many cities close to each other (e.g. the Ruhr Area), district heating areas with an annual demand > 4 TWh are split by NUTS3 boundaries.

The implementation of the district heating area demarcation is done in *DistrictHeatingAreas*, the resulting data is stored in the tables *demand.egon_map_zensus_district_heating_areas* and *demand.egon_district_heating_areas*. The resulting district heating grids for the scenario eGon2035 are visualized in figure district-heating-areas, which also includes a zoom on the district heating grid in Berlin.

The national capacities for each supply technology are taken from the Grid Development Plan (GDP) for the scenario eGon2035, in the eGon100RE scenario they are the result of the *pypsa-eur-sec* run. The distribution of the capacities to district heating grids is done similarly based on [FfE2017], which is also used in the GDP. The basic idea of this method is to use a cascade of heat supply technologies until the heat demand can be covered.

1. Combined heat and power (CHP) plants are assigned to nearby district heating grids first. Their location and thermal capacities are from Marktstammdatenregister [MaStR]. To identify district heating grids that need additional suppliers, the remaining annual heat demand is calculated using the thermal capacities of the CHP plants and assumed full load hours.
2. Large district heating grids with an annual demand that is higher than 96GWh can be supplied by geothermal plants, in case of an intersection of geothermal potential areas and the district heating grid. Smaller district heating grids can be supplied by solar thermal power plants. The national capacities are distributed proportionally to the remaining heat demands. After assigning these plants, the remaining heat demands are reduced by the thermal output and assumed full load hours.
3. Next, the national capacities for central heat pumps and resistive heaters are distributed to all district heating areas proportionally to their remaining demands. Heat pumps are modeled with a time-dependent coefficient of performance depending on the temperature data.
4. In the last step, gas boilers are assigned to every district heating grid regardless of the remaining demand. In the optimization, this can be used as a fall-back option to not run into infeasibilities.

The distribution of CHP plants for different carriers is shown in figure chp-plants.



Hintergrund: Versorgungsbereiche Fernwärme, Berlin

Fig. 9: Defined district heating grids in scenario eGon2035

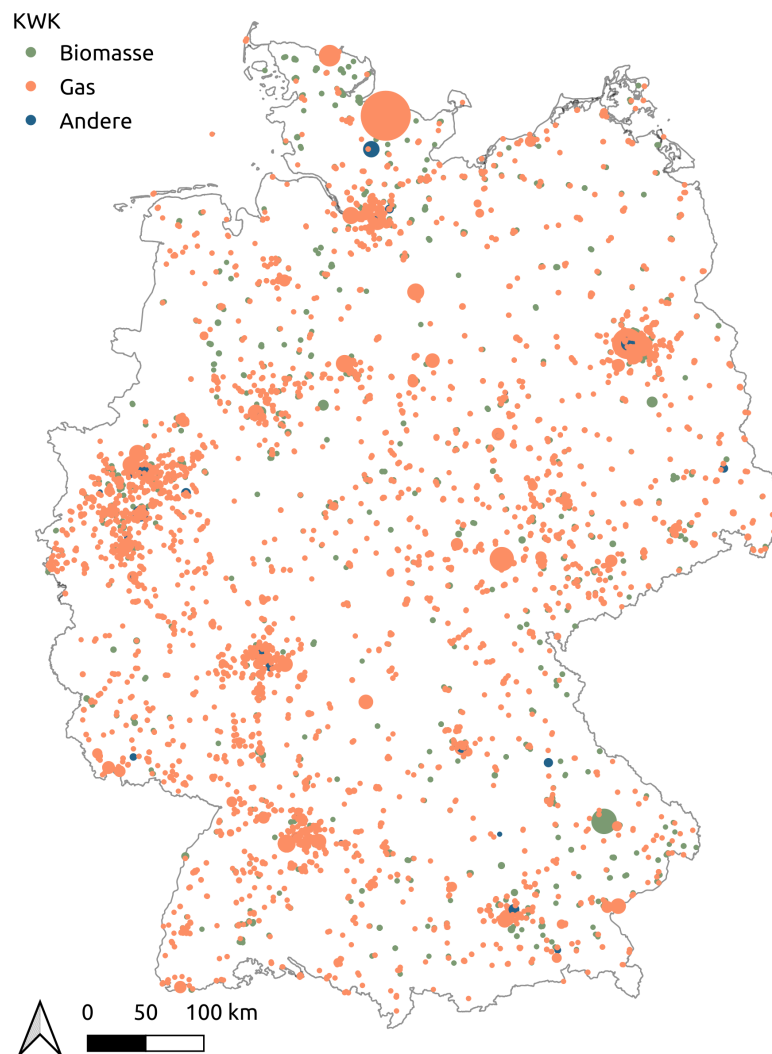


Fig. 10: Spatial distribution of CHP plants in scenario eGon2035

Heat pumps supplying individual buildings are first distributed to each medium-voltage grid district, these capacities are later on further disaggregated to single buildings. Similar to central heat pumps they are modeled with a time-dependent coefficient of performance depending on the temperature data.

The distribution of the national capacities to each medium-voltage grid district is proportional to the heat demand outside of district heating grids.

@RLI: Distribution on building level

All residential and CTS buildings that are neither supplied by a district heating grid nor an individual heat pump are supplied by gas boilers. The demand time series of these buildings are multiplied by the efficiency of gas boilers and aggregated per methane grid node.

All heat supply categories are implemented in the dataset *HeatSupply*. The data is stored in the tables *demand.egon_district_heating* and *demand.egon_individual_heating*.

5.4.3 Gas

Information on gas supply - hydrogen and methane.

5.5 Flexibility options

Different flexibility options are part of the model and can be utilized in the optimization of the energy system. Therefore detailed information about flexibility potentials and their distribution are needed. The considered technologies described in the following chapters range from different storage units, through dynamic line rating to Demand-Side-Management measures.

5.5.1 Demand-Side Management

Demand-side management (DSM) potentials are calculated in function *dsm_cts_ind_processing*. Potentials relevant for the high and extra-high voltage grid are identified in the function *dsm_cts_ind*, potentials within the medium- and low-voltage grids are determined within the function *dsm_cts_ind_individual* in a higher spatial resolution. All this is part of the dataset *DsmPotential*. The implementation is documented in detail within the following student work (in German): [EsterlDentzien].

Loads eligible to be shifted are assumed within industrial loads and loads from Commercial, Trade and Service (CTS). Therefore, load time series from these sectors are used as input data (see section ref:*elec_demand-ref*). Shiftable shares of loads mainly derive from heating and cooling processes and selected energy-intensive industrial processes (cement production, wood pulp, paper production, recycling paper). Technical and sociotechnical constraints are considered using the parametrization elaborated in [Heitkoetter]. An overview over the resulting potentials for scenario eGon2035 can be seen in figure *dsm_potential*. The table below summarizes the aggregated potential for Germany per scenario. As the annual conventional electrical loads are assumed to be lower in the scenario eGon100RE, also the DSM potential decreases compared to the scenario eGon2035.

Table 5: Aggregated DSM Potential for Germany

	CTS	Industry
eGon2035	1.2 GW	150 MW
eGon100RE	900 MW	150 MW

DSM is modelled following the approach of [Kleinhans]. DSM components are created wherever respective loads are seen. Minimum and maximum shiftable power per time step depict time-dependent charging and discharging power of a storage-equivalent buffers. Time-dependent capacities of those buffers account for the time frame of management

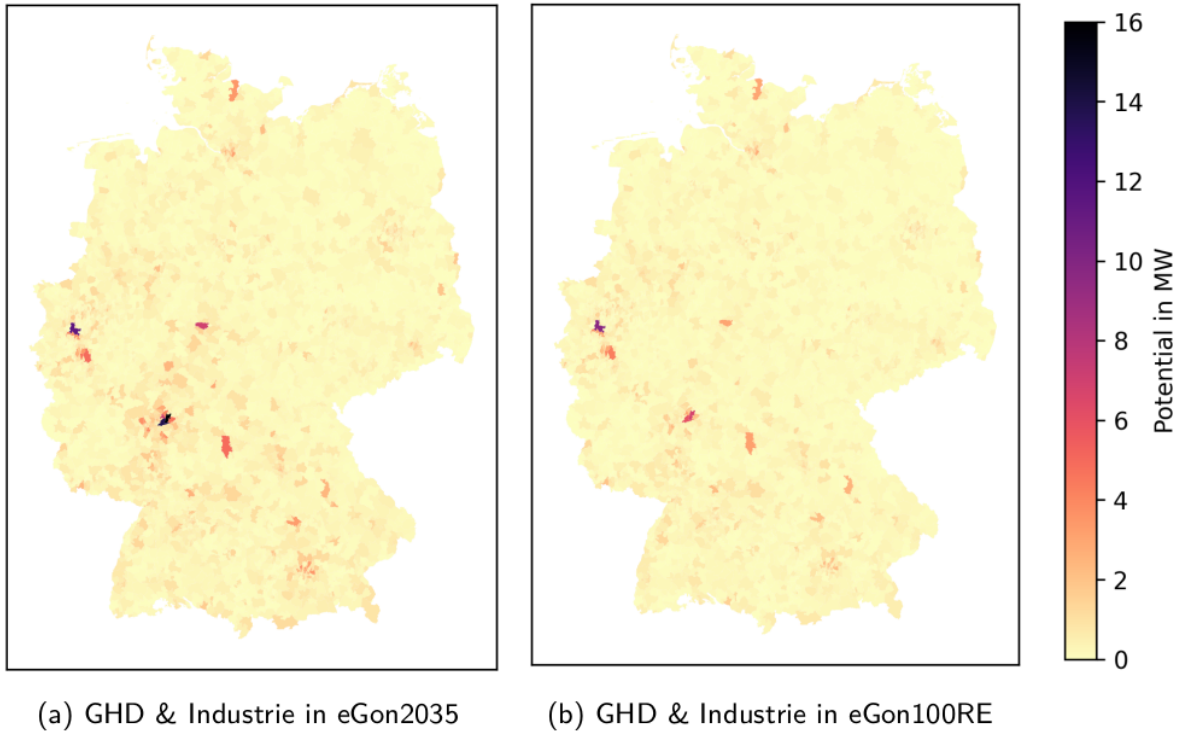


Fig. 11: Aggregated DSM potential in Germany for scenario eGon2035

bounding the period within which the shifting can be conducted. Figure `dsm_shifted_p-example` shows the resulting potential at one exemplary bus.

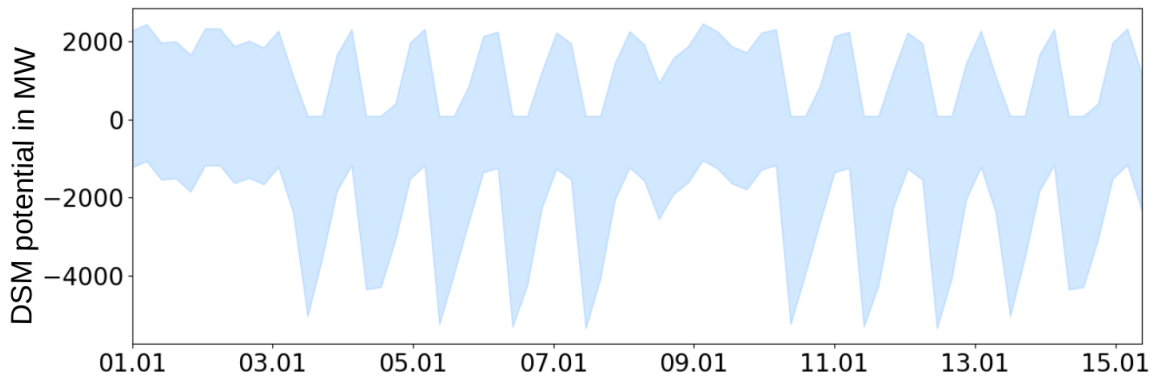


Fig. 12: Time-dependent DSM potential at one exemplary bus

5.5.2 Dynamic line rating

To calculate the transmission capacity of each transmission line in the model, the procedure suggested in the **Principles for the Expansion Planning of the German Transmission Network** [NEP2021] where used:

1. Import the temperature and wind temporal raster layers from ERA-5. Hourly resolution data from the year 2011 was used. Raster resolution latitude-longitude grids at $0.25^\circ \times 0.25^\circ$.

2. Import shape file for the 9 regions proposed by the Principles for the Expansion Planning. See Figure 1.



Figure 1: Representative regions in Germany for DLR analysis [NEP2021]

3. Find the lowest wind speed in each region. To perform this, for each independent region, the wind speed of every cell in the raster layer should be extracted and compared. This procedure is repeated for each hour in the year 2011. The results are the 8760 lowest wind speed per region.
4. Find the highest temperature in each region. To perform this, for each independent region, the temperature of every cell in the raster layer should be extracted and compared. This procedure is repeated for each hour in the year 2011. The results are the 8760 maximum temperature per region.
5. Calculate the maximum capacity for each region using the parameters shown in Figure 2.

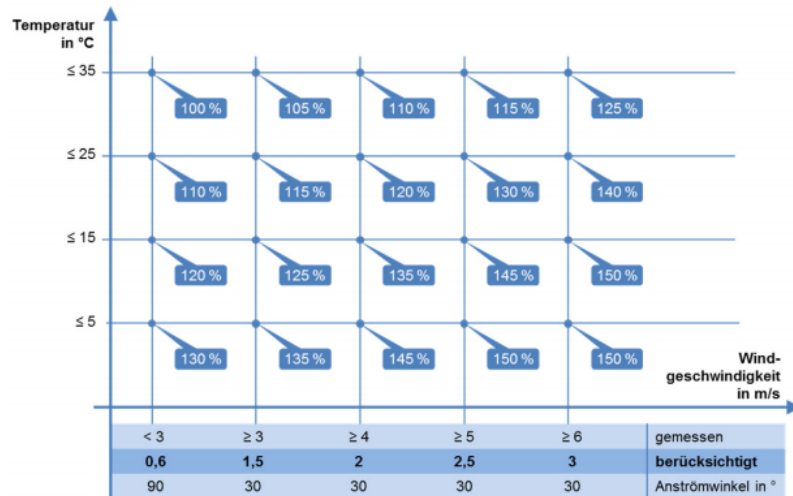


Figure 2: transmission capacity based on max temperature and min wind speed [NEP2021]

6. Assign the maximum capacity of the corresponding region to each transmission line inside each one of them. Crossborder lines and underground lines receive no values. It means that their capacities are static and equal to their nominal values. Lines that cross borders between regions receive the lowest capacity per hour of the regions containing the line.

5.5.3 E-Mobility

What flexibilities does e-mobility provide to the system. How did we implement it?

5.5.4 Battery stores

Battery storage units comprise home batteries and larger, grid-supportive batteries. National capacities for home batteries arise from external sources, e.g. the Grid Development Plan for the scenario eGon2035, whereas the capacities of large-scale batteries are a result of the grid optimization tool eTraGo.

Home battery capacities are first distributed to medium-voltage grid districts (MVGD) and based on that further disaggregated to single buildings. The distribution on MVGD level is done proportional to the installed capacities of solar rooftop power plants, assuming that they are used as solar home storage.

Potential large-scale batteries are included in the data model at every substation. The data model includes technical and economic parameters, such as efficiencies and investment costs. The energy-to-power ratio is set to a fixed value of 6 hours. Other central parameters are given in the following table

Table 6: Parameters of batteries for scenario eGon2035

	Value	Sources
Efficiency store	98 %	[DAE_store]
Efficiency dispatch	98 %	[DAE_store]
Standing loss	0 %	[DAE_store]
Investment costs	838 €/kW	[DAE_store]
Home storage units	16.8 GW	[NEP2021]

On transmission grid level, distinguishing between home batteries and large-scale batteries was not possible. Therefore, the capacities of home batteries were set as a lower boundary of the large-scale battery capacities. This is

implemented in the dataset *StorageEtrago*, the data for batteries in the transmission grid is stored in the database table *grid.egon_etrago_storage*.

5.5.5 Gas stores

Description of methods and assumptions to include potential h2 stores in the system

5.5.6 Hydrogen stores

5.5.7 Methane stores

5.5.8 Heat stores

The heat sector can provide flexibility through stores that allow shifting energy in time. The data model includes hot water tanks as heat stores in individual buildings and pit thermal energy storage for district heating grids (further described in district-heating).

Within the data model, potential locations as well as technical and economic parameters of these stores are defined. The installed store and (dis-)charging capacities are part of the grid optimization methods that can be performed by eTraGo. The power-to-energy ratio is not predefined but a result of the optimization, which allows to build heat stores with various time horizons.

Individual heat stores can be built in every building with an individual heat pump. Central heat stores can be built next to district heating grids. There are no maximum limits for the energy output as well as (dis-)charging capacities implemented yet.

Central cost assumptions for central and decentral heat stores are listed in the table below. The parameters can differ for each scenario in order to include technology updates and learning curves. The table focuses on the scenario eGon2035.

Table 7: Parameters of heat stores

	Technology	Costs for store capacity	Costs for (dis-)charging capacity	Round-trip efficiency	Sources
District heating	Pit thermal energy storage	0.51 EUR / kWh	0 EUR / kW	70 %	[DAE_store]
Buildings with heat pump	Water tank	1.84 EUR / kWh	0 EUR / kW	70 %	[DAE_store]

The heat stores are implemented as a part of the dataset *HeatEtrago*, the data is written into the tables *grid.egon_etrago_bus*, *grid.egon_etrago_link* and *grid.egon_etrago_store*.

5.6 Published data

CHAPTER 6

Literature

The research project eGo_n and egon-data are collaborative projects with several people contributing to it. The following section gives an overview of applicable guidelines and rules to enable a prospering collaboration. Any external contributions are welcome as well, and they are greatly appreciated! Every little bit helps, and credit will always be given.

7.1 Bug reports and feature requests

The best way to report bugs, inform about intended developments, send feedback or propose a feature is to file an issue at <https://github.com/openego/eGon-data/issues>.

Please tag your issue with one of the predefined labels as it helps others to keep track of unsolved bugs, open tasks and questions.

To inform others about intended developments please include: * a description of the purpose and the value it adds * outline the required steps for implementation * list open questions

When reporting a bug please include all information needed to reproduce the bug you found. This may include information on

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

7.2 Contribution guidelines

7.2.1 Development

Adding changes to the `egon-data` repository should follow some guidelines:

1. Create an [issue](#) in our [repository](#) to describe the intended developments briefly
2. Create a branch for your issue related development from the `dev-branch` following our branch naming convention:

```
git checkout -b `<prefix>/#<issue-id>-very-brief-description`
```

where *issue-id* is the issue number on GitHub and *prefix* is one of

- features
- fixes
- refactorings

depending on which one is appropriate. This command creates a new branch in your local repository, in which you can now make your changes. Be sure to check out our [style conventions](#) so that your code is in line with them. If you don't have push rights to our [repository](#), you need to fork it via the "Fork" button in the upper right of the [repository](#) page and work on the fork.

3. Make sure to update the documentation along with your code changes
4. When you're done making changes run all the checks and docs builder with `tox` one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add -p
git commit
git push origin features/#<issue-id>-very-brief-description
```

Note that the `-p` switch will make `git add` iterate through your changes and prompt for each one on whether you want to include it in the upcoming commit. This is useful if you made multiple changes which should conceptually be grouped into different commits, like e.g. fixing the documentation of one function and changing the implementation of an unrelated one in parallel, because it allows you to still make separate commits for these changes. It has the drawback of not picking up new files though, so if you added files and want to put them under version control, you have to add them explicitly by running `git add FILE1 FILE2 ...` instead.

6. Submit a pull request through the GitHub website.

7.2.2 Code and Commit Style

We try to adhere to the [PEP 8 Style Guide](#) wherever possible. In addition to that, we use a *code formatter* to have a consistent style, even in cases where PEP 8 leaves multiple degrees of freedom. So please run your code through `black` before committing it.¹ PEP 8 also specifies a way to group imports, onto which we put the additional constraint that the imports within each group are ordered alphabetically. Once again, you don't have to keep track of this manually, but you can use `isort` to have imports sorted automatically. Note that *pre-commit* hooks are configured

¹ If you want to be really nice, run any file you touch through `black` before making changes, and commit the result separately from other changes.. The repository may contain wrongly formatted legacy code, and this way you commit eventually necessary style fixes separated from your actually meaningful changes, which makes the reviewers job a lot easier.

for this repository, so you can just `pip install pre-commit` followed by `pre-commit install` in the repository, and every commit will automatically be checked for style violations.

Unfortunately these tools don't catch everything, so here's a short list of things you have to keep track of manually:

- `Black` can't automatically break up overly long strings, so make use of Python's automatic string concatenation feature by e.g. converting

```
something = "A really really long string"
```

into the equivalent:

```
something = (
    "A really really"
    " long string"
)
```

- `Black` also can't check whether you're using readable names for your variables. So please don't use abbreviations. Use [readable names](#).
- `Black` also can't reformat your comments. So please keep in mind that PEP 8 specifies a line length of 72 for free flowing text like comments and docstrings. This also extends to the documentation in reStructuredText files.

Last but not least, commit messages are a kind of documentation, too, which should adhere to a certain style. There are quite a few documents detailing this style, but the shortest and easiest to find is probably <https://commit.style>. If you have 15 minutes instead of only five to spare, there's also a very good and only [slightly longer article](#) on this subject, containing references to other style guides, and also explaining why commit messages are important.

At the very least, try to only commit small, related changes. If you have to use an "and" when trying to summarize your changes, they should probably be grouped into separate commits.

7.2.3 Pull Request Guidelines

We use pull requests (PR) to integrate code changes from branches. PRs always need to be reviewed (exception proves the rule!). Therefore, ask one of the other developers for reviewing your changes. Once approved, the PR can be merged. Please delete the branch after merging.

Before requesting a review, please

1. Include passing tests (`run tox`).²
2. Let the workflow run in *Test mode* once from scratch to verify successful execution
3. Make sure that your changes are tested in integration with other tasks and on a complete run at least once by merging them into the [continuous-integration/run-everything-over-the-weekend](#) branch. This branch will regularly be checked out and tested on a complete workflow run on friday evening.
4. Update documentation when there's new API, functionality etc.
5. Add a note to `CHANGELOG.rst` about the changes and refer to the corresponding Github issue.
6. Add yourself to `AUTHORS.rst`.

When requesting reviews, please keep in mind it might be a significant effort to review the PR. Try to make it easier for them and keep the overall effort as low as possible. Therefore,

- asking for reviewing specific aspects helps reviewers a lot to focus on the relevant parts

² If you don't have all the necessary Python versions available locally you can rely on CI via GitHub actions - it will [run the tests](#) for each change you add in the pull request.

It will be slower though ...

- when multiple people are asked for a review it should be avoided that they check/test the same things. Be even more specific what you expect from someone in particular.

7.2.4 What needs to be reviewed?

Things that definitely should be checked during a review of a PR:

- *Is the code working?* The contributor should already have made sure that this is the case. Either by automated test or manual execution.
- *Is the data correct?* Verifying that newly integrated and processed data is correct is usually not possible during reviewing a PR. If it is necessary, please ask the reviewer specifically for this.
- *Do tests pass?* See automatic checks.
- *Is the documentation up-to-date?* Please check this.
- *Was CHANGELOG.rst updated accordingly?* Should be the case, please verify.
- *Is metadata complete and correct (in case of data integration)?* Please verify. In case of a pending metadata creation make sure an appropriate issue is filed.

7.3 Extending the data workflow

The egon-data workflow uses Apache Airflow which organizes the order of different processing steps and their execution.

7.3.1 How to add Python scripts

To integrate a new Python function to the egon-data workflow follow the steps listed:

1. Add your well documented script to the egon-data repository
2. Integrate functions which need to be called within the workflow to pipeline.py, which organizes and calls the different tasks within the workflow
3. Define the interdependencies between the scripts by setting the task downstream to another required task
4. The workflow can now be triggered via Apache Airflow

7.3.2 Where to save (downloaded) data?

If a task requires to retrieve some data from external sources which needs to be saved locally, please use *CWD* to store the data. This is achieved by using

```
from pathlib import Path
from urllib.request import urlretrieve

filepath = Path(".") / "filename.csv"
urlretrieve("https://url/to/file", filepath)
```

7.3.3 Add metadata

Add a metadata for every dataset you create for describing data with machine-readable information. Adhere to the OEP Metadata v1.4.1, you can follow [the example](#) to understand how the fields are used. Field are described in detail in the [Open Energy Metadata Description](#).

You can obtain the metadata string from a table you created in SQL via

```
SELECT obj_description('<SCHEMA>.<TABLE>'::regclass);
```

Alternatively, you can write the table comment directly to a JSON file by

```
psql -h <HOST> -p <PORT> -d <DB> -U <USER> -c "\COPY (SELECT obj_description('<SCHEMA>
↪.<TABLE>'::regclass)) TO '/PATH/TO/FILE.json';"
```

For bulk export of all DB's table comments you can use [this script](#). Please verify that your metadata string is in compliance with the OEP Metadata standard version 1.4.1 using the [OMI tool](#) (tool is shipped with eGon-data):

```
omi translate -f oep-v1.4 -t oep-v1.4 metadata_file.json
```

If your metadata string is correct, OMI puts the keys in the correct order and prints the full string (use `-o` option for export).

You may omit the fields *id* and *publicationDate* in your string as it will be automatically set at the end of the pipeline but you're required to set them to some value for a complete validation with OMI. For datasets published on the OEP *id* will be the URL which points to the table, it will follow the pattern <https://openenergy-platform.org/dataedit/view/SCHEMA/TABLE>.

For previous discussions on metadata, you may want to check [PR 176](#).

Helpers

You can use the [Metadata creator GUI](#). Fill the fields and hit *Edit JSON* to get the metadata string. Vice versa, you can paste a metadata string into this box and the fields will be filled automatically which may be helpful if you want to amend existing strings.

There are some **licence templates** provided in `egon.data.metadata` you can make use of for fields 11.4 and 12 of the [Open Energy Metadata Description](#). Also, there's a template for the **metaMetadata** (field 16).

There are some functions to quickly generate a template for the **resource fields** (field 14.6.1 in [Open Energy Metadata Description](#)) from a SQLA table class or a DB table. This might be especially helpful if your table has plenty of columns.

- From SQLA table class: `egon.data.metadata.generate_resource_fields_from_sqla_model()`
- From database table: `egon.data.metadata.generate_resource_fields_from_db_table()`

Sources

The **sources** (field 11) are the most important parts of the metadata which need to be filled manually. You may also add references to tables in eGon-data (e.g. from an upstream task) so you don't have to list all original sources again. Make sure you include all upstream attribution requirements.

The following example uses various input datasets whose attribution must be retained:

```
"sources": [
  {
    "title": "eGo^n - Medium voltage grid districts",
    "description": (
      "Medium-voltage grid districts describe the area supplied by "
      "one MV grid. Medium-voltage grid districts are defined by one "
      "polygon that represents the supply area. Each MV grid district "
      "is connected to the HV grid via a single substation."
    ),
    "path": "https://openenergy-platform.org/dataedit/view/"
           "grid/egon_mv_grid_district", # "id" in the source dataset
    "licenses": [
      license_odbl(attribution=
        "© OpenStreetMap contributors, 2021; "
        "© Statistische Ämter des Bundes und der Länder, 2014; "
        "© Statistisches Bundesamt, Wiesbaden 2015; "
        "(Daten verändert) "
      )
    ]
  },
  # more sources...
]
```

7.3.4 Adjusting test mode data

When integrating new data or data processing scripts, make sure the *Test mode* still works correctly on a limited subset of data. In particular, if a new external data sources gets integrated make sure the data gets cut to the region of the test mode.

7.4 Documentation

eGon-data could always use more documentation, whether as part of the official eGon-data docs, in docstrings, or even in articles, blog posts or similar resources. Always keep in mind to update the documentation along with your code changes though.

The changes of the documentation in a feature branch get visible once a pull request is opened.

7.4.1 How to document Python scripts

Use docstrings to document your Python code. Note that PEP 8 also contains a [section](#) on docstrings and that there is a whole [PEP](#) dedicated to docstring conventions. Try to adhere to both of them. Additionally every Python script needs to contain a header describing the general functionality and objective and including information on copyright, license and authors.

```
""" Provide an example of the first line of a module docstring.

This is an example header describing the functionalities of a Python
script to give the user a general overview of what's happening here.
"""

__copyright__ = "Example Institut"
__license__ = "GNU Affero General Public License Version 3 (AGPL-3.0)"
```

(continues on next page)

(continued from previous page)

```
__url__ = "https://github.com/openego/eGon-data/blob/main/LICENSE"
__author__ = "github_alias1, github_alias2"
```

7.4.2 How to document SQL scripts

Please also add a similar header to your SQL scripts to give users and fellow developers an insight into your scripts and the methodologies applied. Please describe the content and objectives of the script briefly but as detailed as needed to allow other to comprehend how it works.

```
/*
This is an example header describing the functionalities of a SQL
script to give the user a general overview what's happening here

__copyright__ = "Example Institut"
__license__ = "GNU Affero General Public License Version 3 (AGPL-3.0)"
__url__ = "https://github.com/openego/eGon-data/blob/main/LICENSE"
__author__ = "github_alias1, github_alias2"
*/
```

You can build the documentation locally with (executed in the repos root directory)

```
sphinx-build -E -a docs docs/_build/
```

Eventually, you might need to install additional dependencies for building the documentation:

```
pip install -r docs/requirements.txt
```

7.4.3 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel*:

```
tox -p auto
```


CHAPTER 8

Authors

- Guido Pleßmann, Ilka Cußmann, Stephan Günther, Jonathan Amme, Julian Endres, Kilian Helfenbein - <https://github.com/openego/eGon-data>

9.1 Unreleased

9.1.1 Added

- Include description of the egon-data workflow in our documentation [#23](#)
- There's now a wrapper around *subprocess.run* in *egon.data.subprocess.run*. This wrapper catches errors better and displays better error messages than Python's built-in function. Use this wrapper wenn calling other programs in Airflow tasks.
- You can now override the default database configuration using command line arguments. Look for the switches starting with `--database` in `egon-data --help`. See [PR #159](#) for more details.
- Docker will not be used if there is already a service listening on the HOST:PORT combination configured for the database.
- You can now supply values for the command line arguments for `egon-data` using a configuration file. If the configuration file doesn't exist, it will be created by `egon-data` on it's first run. Note that the configuration file is read from and written to the directory in which `egon-data` is started, so it's probably best to run `egon-data` in a dedicated directory. There's also the new function *egon.data.config.settings* which returns the current configuration settings. See [PR #159](#) for more details.
- You can now use tasks which are not part of a `Dataset`, i.e. which are unversioned, as dependencies of a dataset. See [‘PR #318’](#) for more details.
- You can now force the tasks of a `Dataset` to be always executed by giving the version of the `Dataset` a `".dev"` suffix. See [‘PR #318’](#) for more details.
- OSM data import as done in `open_ego #1` which was updated to the latest long-term data set of the 2021-01-01 in [#223](#)
- Verwaltungsgebiete data import (vg250) more or less done as in `open_ego #3`
- Zensus population data import [#2](#)
- Zensus data import for households, apartments and buildings [#91](#)

- DemandRegio data import for annual electricity demands #5
- Download cleaned open-MaStR data from Zenodo #14
- NEP 2021 input data import #45
- Option for running workflow in test mode #112
- Abstraction of hvmv and ehv substations #9
- Filter zensus being inside Germany and assign population to municipalities #7
- RE potential areas data import #124
- Heat demand data import #101
- Demographic change integration #47
- Creation of voronoi polygons for hvmv and ehv substations #9
- Add hydro and biomass power plants eGon2035 #127
- Creation of the ehv/hv grid model with osmTGmod, see [issue #4](#) and [PR #164](#)
- Identification of medium-voltage grid districts #10
- Distribute electrical demands of households to zensus cells #181
- Distribute electrical demands of cts to zensus cells #210
- Include industrial sites' download, import and merge #117
- Integrate scenario table with parameters for each sector #177
- The volume of the docker container for the PostgreSQL database is saved in the project directory under *docker/database-data*. The current user (\$USER) is owner of the volume. Containers created prior to this change will fail when using the changed code. The container needs to be re-created. #228
- Extract landuse areas from OSM #214
- Integrate weather data and renewable feedin timeseries #19
- Create and import district heating areas #162
- Integrate electrical load time series for cts sector #109
- Assign voltage level and bus_id to power plants #15
- Integrate solar rooftop for etrago tables #255
- Integrate gas bus and link tables #198
- Integrate data bundle #272
- Add household electricity demand time series, mapping of demand profiles to census cells and aggregated household electricity demand time series at MV grid district level #256
- Integrate power-to-gas installation potential links #293
- Integrate distribution of wind onshore and pv ground mounted generation #146
- Integrate dynamic line rating potentials #72
- Integrate gas voronoi polygons #308
- Integrate supply strategies for individual and district heating #232
- Integrate gas production #321
- Integrate industrial time series creation #237

- Merge electrical loads per bus and export to etrago tables #328
- Insert industrial gas demand #358
- Integrate existing CHP and extended CHP > 10MW_el #266
- Add random seed to CLI parameters #351
- Extend zensus by a combined table with all cells where there's either building, apartment or population data #359
- Include allocation of pumped hydro units #332
- Add example metadata for OSM, VG250 and Zensus VG250. Add metadata templates for licences, context and some helper functions. Extend docs on how to create metadata for tables. #139
- Integrate DSM potentials for CTS and industry #259
- Assign weather cell id to weather dependant power plants #330
- Distribute wind offshore capacities #329
- Add CH4 storages #405
- Include allocation of conventional (non CHP) power plants #392
- Fill egon-etrago-generators table #485
- Include time-dependent coefficient of performance for heat pumps #532
- Limit number of parallel processes per task #265
- Include biomass CHP plants to eTraGo tables #498
- Include Pypsa default values in table creation #544
- Include PHS in eTraGo tables #333
- Include feedin time series for wind offshore #531
- Include carrier names in eTraGo table #551
- Include hydrogen infrastructure for eGon2035 scenario #474
- Include downloaded pypsa-eur-sec results #138
- Create heat buses for eGon100RE scenario #582
- Filter for DE in gas infrastructure deletion at beginning of respective tasks #567
- Insert open cycle gas turbines into eTraGo tables #548
- Preprocess buildings and amenities for LV grids #262
- Assign household profiles to OSM buildings #435
- Add link to meta creator to docs #599
- Add extendable batteries and heat stores #566
- Add efficiency, capital_cost and marginal_cost to gas related data in etrago tables #596
- Add wind onshore farms for the eGon100RE scenario #690
- The shared memory under `"/dev/shm"` is now shared between host and container. This was done because Docker has a rather tiny default for the size of `"/dev/shm"` which caused random problems. Guessing what size is correct is also not a good idea, so sharing between host and container seems like the best option. This restricts using *egon-data* with docker to Linux and MacOS, if the latter has `"/dev/shm"` but seems like the best course of action for now. Done via PR #703 and hopefully prevents issues #702 and #267 from ever occurring again.

- Provide wrapper to catch DB unique violation [#514](#)
- Add electric scenario parameters for eGon100RE [#699](#)
- Introduce Sanity checks for eGon2035 [#382](#)
- Add motorized individual travel [#553](#)
- Allocating MaStR PV rooftop power plants to OSM and synthetic buildings. Desaggregating PV rooftop scenarios to mv grid districts and OSM and synthetic buildings. [#684](#)
- Add mapping zensus - weather cells [#845](#)
- Add pv rooftop plants per mv grid for eGon100RE [#861](#)
- Integrated heavy duty transport FCEV [#552](#)
- Assign CTS demands to buildings [#671](#)
- Add sanity checks for residential electricity loads [#902](#)
- Add sanity checks for cts loads [#919](#)
- Add distribution of CHP plants for eGon100RE [#851](#)
- Add mapping table for all used buildings [#962](#)
- Add charging infrastructure for e-mobility [#937](#)
- Add zipfile check [#969](#)
- Add marginal costs for generators abroad and for carriers nuclear and coal [#907](#)
- Add wind off shore power plants for eGon100RE [#868](#)
- Write simBEV metadata to DB table [PR #978](#)
- Add voltage level for electricity building loads [#955](#)
- Add disaggregation of pv home batteries onto buildings [#988](#)
- Desaggregation of DSM time series onto CTS consumers per bus id and individual industry consumers. [#1048](#)
- Add load areas [#1014](#)
- Add new MaStR dataset [#1051](#)
- Heat pump disaggregation to buildings [PR #903](#)
- Add low flex scenario 'eGon2035_lowflex' [#822](#)
- Add MaStR geocoding and handling of conventional generators [#1095](#)

9.1.2 Changed

- Adapt structure of the documentation to project specific requirements [#20](#)
- Switch from Travis to GitHub actions for CI jobs [#92](#)
- Rename columns to id and zensus_population_id in zensus tables [#140](#)
- Revise docs CONTRIBUTING section and in particular PR guidelines [#88](#) and [#145](#)
- Drop support for Python 3.6 [#148](#)
- Improve selection of zensus data in test mode [#151](#)
- Delete tables before re-creation and data insertion [#166](#)

- Adjust residential heat demand in unpopulated zenus cells #167
- Introduce mapping between VG250 municipalities and census cells #165
- Delete tables if they exist before re-creation and data insertation #166
- Add gdal to pre-requisites #185
- Update task zensus-inside-germany #196
- Update installation of demandregio's disaggregator #202
- Update etrago tables #243 and #285
- Migrate VG250 to datasets #283
- Allow configuring the airflow port #281
- Migrate mastr, mv_grid_districts and re_potential_areas to datasets #297
- Migrate industrial sites to datasets #237
- Rename etrago tables from e.g. egon_pf_hv_bus to egon_etrago bus etc. #334
- Move functions used by multiple datasets #323
- Migrate scenario tables to datasets #309
- Migrate weather data and power plants to datasets #314
- Create and fill table for CTS electricity demand per bus #326
- Migrate osmTGmod to datasets #305
- Filter osm landuse areas, rename industrial sites tables and update load curve function #378
- Remove version columns from eTraGo tables and related code #384
- Remove country column from scenario capacities table #391
- Update version of zenodo download #397
- Rename columns gid to id #169
- Remove upper version limit of pandas #383
- Use random seed from CLI parameters for CHP and society prognosis functions #351
- Changed demand.egon_schmidt_industrial_sites - table and merged table (industrial_sites) #423
- Replace 'gas' carrier with 'CH4' and 'H2' carriers #436
- Adjust file path for industrial sites import #418
- Rename columns subst_id to bus_id #335
- Apply black and isort for all python scripts #463
- Update deposit id for zenodo download #498
- Add to etrago.setug.py the busmap table #484
- Migrate dlr script to datasets #508
- Migrate loadarea scripts to datasets #525
- Migrate plot.py to dataset of district heating areas #527
- Migrate substation scripts to datasets #304
- Update deposit_id for zenodo download #540

- Add household demand profiles to etrago table #381
- Migrate zensus scripts to datasets #422
- Add information on plz, city and federal state to data on mastr without chp #425
- Assign residential heat demands to osm buildings #557
- Add foreign gas buses and adjust cross bordering pipelines #545
- Integrate fuel and CO2 costs for eGon2035 to scenario parameters #549
- Aggregate generators and stores for CH4 #629
- Fill missing household data for populated cells #431
- Fix RE potential areas outside of Germany by updating the dataset. Import files from data bundle. #592 #595
- Add DC lines from Germany to Sweden and Denmark #611
- H2 demand is met from the H2_grid buses. In Addition, it can be met from the H2_saltcavern buses if a proximity criterion is fulfilled #620
- Create H2 pipeline infrastructure for eGon100RE #638
- Change refinement method for households types #651
- H2 feed in links are changed to non extendable #653
- Remove the ‘_fixed’ suffix #628
- Fill table demand.egon_demandregio_zensus_electricity after profile allocation #586
- Change method of building assignment #663
- Create new OSM residential building table #587
- Move python-operators out of pipeline #644
- Add annualized investment costs to eTraGo tables #672
- Improve modelling of NG and biomethane production #678
- Unify carrier names for both scenarios #575
- Add automatic filtering of gas data: Pipelines of length zero and gas buses isolated of the grid are deleted. #590
- Add gas data in neighbouring countries #727
- Aggregate DSM components per substation #661
- Aggregate NUTS3 industrial loads for CH4 and H2 #452
- Update OSM dataset from 2021-02-02 to 2022-01-01 #486
- Update deposit id to access v0.6 of the zenodo repository #627
- Include electricity storages for eGon100RE scenario #581
- Update deposit id to access v0.7 of the zenodo repository #736
- Include simplified restrictions for H2 feed-in into CH4 grid #790
- Update hh electricity profiles #735
- Improve CH4 stores and productions aggregation by removing dedicated task #775
- Add CH4 stores in Germany for eGon100RE #779
- Assignment of H2 and CH4 capacities for pipelines in eGon100RE #686

- Update deposit id to access v0.8 of the zenodo repository #760
- Add primary key to table openstreetmap.osm_ways_with_segments #787
- Update pypsa-eur-sec fork and store national demand time series #402
- Move and merge the two assign_gas_bus_id functions to a central place #797
- Add coordinates to non AC buses abroad in eGon100RE #803
- Integrate additional industrial electricity demands for eGon100RE #817
- Set non extendable gas components from p-e-s as so for eGon100RE #877
- Integrate new data bundle using zenodo sandbox #866
- Add noflex scenario for motorized individual travel #821
- Allocate PV home batteries to mv grid districts #749
- Add sanity checks for motorized individual travel #820
- Parallelize sanity checks #882
- Insert crossboarding gas pipeline with Germany in eGon100RE #881
- Harmonize H2 carrier names in eGon100RE #929
- Rename noflex to lowflex scenario for motorized individual travel #921
- Update creation of heat demand timeseries #857 #856
- Overwrite retrofitted_CH4pipeline-to-H2pipeline_share with pes result #933
- Adjust H2 industry profiles abroad for eGon2035 #940
- Introduce carrier name 'others' #819
- Add rural heat pumps per medium voltage grid district #987
- Add eGon2021 scenario to demandregio dataset #1035
- Update MaStR dataset #519
- Add missing VOM costs for heat sector components #942
- Add sanity checks for gas sector in eGon2035 #864
- Desaggregate industry demands to OSM areas and industrial sites #1001
- Add gas generator in Norway #1074
- SQLAlchemy *engine* objects created via `egon.data.db.engine` are now cached on a per process basis, so only one *engine* is ever created for a single process. This fixes issue #799.
- Insert rural heat per supply technology #1026
- Insert lifetime for components from p-e-s in eGon100RE #1073
- Change hgv data source to use database #1086
- Change desposit ID for data_bundle download from zenodo sandbox #1110
- Use MaStR geocoding results for pv rooftop to buildings mapping workflow #1095
- Rename eMob MIT carrier names (use underscores) #1105

9.1.3 Bug Fixes

- Some dependencies have their upper versions restricted now. This is mostly due to us not yet supporting Airflow 2.0 which means that it will no longer work with certain packages, but we also won't get an upper version limit for those from Airflow because version 1.X is unlikely to get an update. So we had to make some implicit dependencies explicit in order to give them their upper version limits. Done via [PR #692](#) in order to fix issues [#343](#), [#556](#), [#641](#) and [#669](#).
- Heat demand data import [#157](#)
- Substation sequence [#171](#)
- Adjust names of demandregions nuts3 regions according to nuts version 2016 [#201](#)
- Delete zensus buildings, apartments and households in unpopulated cells [#202](#)
- Fix input table of electrical-demands-zensus [#217](#)
- Import heat demand raster files successively to fix import for dataset==Everything [#204](#)
- Replace wrong table name in SQL function used in substation extraction [#236](#)
- Fix osmtgmod for osm data from 2021 by updating substation in Garenfeld and set srid [#241](#) [#258](#)
- Adjust format of voltage levels in hvmv substation [#248](#)
- Change order of osmtgmod tasks [#253](#)
- Fix missing municipalities [#279](#)
- Fix import of hydro power plants [#270](#)
- Fix path to osm-file for osmtgmod_osm_import [#258](#)
- Fix conflicting docker containers by setting a project name [#289](#)
- Update task insert-nep-data for pandas version 1.3.0 [#322](#)
- Fix versioning conflict with mv_grid_districts [#340](#)
- Set current working directory as java's temp dir when executing osmosis [#344](#)
- Fix border gas voronoi polygons which had no bus_id [#362](#)
- Add dependency from WeatherData to Vg250 [#387](#)
- Fix unnecessary columns in normal mode for inserting the gas production [#390](#)
- Add xlrd and openpyxl to installation setup [#400](#)
- Store files of OSM, zensus and VG250 in working dir [#341](#)
- Remove hard-coded slashes in file paths to ensure Windows compatibility [#398](#)
- Add missing dependency in pipeline.py [#412](#)
- Add prefix egon to MV grid district tables [#349](#)
- Bump MV grid district version no [#432](#)
- Add curl to prerequisites in the docs [#440](#)
- Replace NAN by 0 to avoid empty p_set column in DB [#414](#)
- Exchange bus 0 and bus 1 in Power-to-H2 links [#458](#)
- Fix missing cts demands for eGon2035 [#511](#)
- Add *data_bundle* to *industrial_sites* task dependencies [#468](#)

- Lift *geopandas* minimum requirement to 0.10.0 #504
- Use inbuilt *datetime* package instead of *pandas.datetime* #516
- Add missing ‘sign’ for CH4 and H2 loads #538
- Delete only AC loads for eTraGo in *electricity_demand_etrigo* #535
- Filter target values by scenario name #570
- Reduce number of timesteps of hh electricity demand profiles to 8760 #593
- Fix assignemnt of heat demand profiles at German borders #585
- Change source for H2 steel tank storage to Danish Energy Agency #605
- Change carrier name from ‘pv’ to ‘solar’ in *eTraGo_generators* #617
- Assign “carrier” to transmission lines with no value in *grid.egon_etrigo_line* #625
- Fix deleting from eTraGo tables #613
- Fix positions of the foreign gas buses #618
- Create and fill *transfer_busses* table in *substation-dataset* #610
- H2 steel tanks are removed again from saltcavern storage #621
- Timeseries not deleted from *grid.etrigo_generator_timeseries* #645
- Fix function to get scaled hh profiles #674
- Change order of *pypsa-eur-sec* and *scenario-capacities* #589
- Fix gas storages capacities #676
- Distribute rural heat supply to *residetntial* and *service demands* #679
- Fix time series creation for pv rooftop #688
- Fix extraction of buildings without amenities #693
- Assign DLR capacities to every transmission line #683
- Fix solar ground mounted total installed capacity #695
- Fix twisted number error residential demand #704
- Fix industrial H2 and CH4 demand for eGon100RE scenario #687
- Clean up “*pipeline.py*” #562
- Assign timeseries data to crossborder generators *ego2035* #724
- Add missing dataset dependencies in “*pipeline.py*” #725
- Fix assignemnt of impedances (x) to *etrigo* tables #710
- Fix *country_code* attribution of two gas buses #744
- Fix *voronoi* assignemnt for enclaves #734
- Set lengths of non-pipeline links to 0 #741
- Change table name from *boundaries.saltstructures_inspee* to *boundaries.inspee_saltstructures* #746
- Add missing marginal costs for conventional generators in Germany #722
- Fix carrier name for solar ground mounted in scenario parameters #752

- Create rural_heat buses only for mv grid districts with heat load #708
- Solve problem while creating generators series data egon2035 #758
- Correct wrong carrier name when assigning marginal costs #766
- Use db.next_etrago_id in dsm and pv_rooftop dataset #748
- Add missing dependency to heat_etrago #771
- Fix country code of gas pipeline DE-AT #813
- Fix distribution of resistive heaters in district heating grids #783
- Fix missing reservoir and run_of_river power plants in eTraGo tables, Modify fill_etrago_gen to also group generators from eGon100RE, Use db.next_etrago_id in fill_etrago_gen #798 #776
- Fix model load timeseries in motorized individual travel #830
- Fix gas costs #847
- Add imports that have been wrongly deleted #849
- Fix final demand of heat demand timeseries #781
- Add extendable batteries only to buses at substations #852
- Move class definition for grid.egon_gas_voronoi out of etrago_setup #888
- Temporarily set upper version limit for pandas #829
- Change industrial gas load modelling #871
- Delete eMob MIT data from eTraGo tables on init #878
- Fix model id issues in DSM potentials for CTS and industry #901
- Drop isolated buses and transformers in eHV grid #874
- Model gas turbines always as links #914
- Drop era5 weather cell table using cascade #909
- Remove drop of p_set and q_set for loads without timeserie #971
- Delete gas bus with wrong country code #958
- Overwrite capacities for conventional power plants with data from nep list #403
- Make gas grid links bidirectional #1021
- Correct gas technology costs for eGon100RE #984
- Adjust p_nom and marginal cost for OCGT in eGon2035 #863
- Mismatch of building bus_ids from cts_heat_demand_building_share and mapping table #989
- Fix census weather cells mapping #1031
- Fix solar rooftop in test mode #1055
- Add missing filter for scenario name in chp expansion #1015
- Fix installed capacity per individual heat pump #1058
- Add missing gas turbines abroad #1079
- Fix gas generators abroad (marginal cost and e_nom_max) #1075
- Fix gas pipelines isolated of the German grid #1081

- Fix aggregation of DSM-components [#1069](#)
- Fix URL of TYNDP scenario dataset
- Automatically generated tasks now get unique `task_ids`. Fixes issue [#985](#) via PR [#986](#).
- Adjust capacities of German CH4 stores [#1096](#)
- Fix faulty DSM time series [#1088](#)
- Set upper limit on commissioning date for units from MaStR dataset [#1098](#)
- Fix conversion factor for CH4 loads abroad in eGon2035 [#1104](#)
- Change structure of documentation in rtd [#11126](#)

echo (*message*)

10.1 airflow

10.2 cli

Module that contains the command line app.

Why does this file exist, and why not put this in `__main__`?

You might be tempted to import things from `__main__` later, but that will cause problems: the code will get executed twice:

- When you run `python -megon.data` python will execute `__main__.py` as a script. That means there won't be any `egon.data.__main__` in `sys.modules`.
- When you import `__main__` it will get executed again (as a module) because there's no `egon.data.__main__` in `sys.modules`.

Also see (1) from <http://click.pocoo.org/5/setuptools/#setuptools-integration>

main ()

10.3 config

datasets (*config_file=None*)

Return dataset configuration.

Parameters **config_file** (*str, optional*) – Path of the dataset configuration file in YAML format. If not supplied, a default configuration shipped with this package is used.

Returns *dict* – A nested dictionary containing the configuration as parsed from the supplied file, or the default configuration if no file was given.

paths (*pid=None*)

Obtain configuration file paths.

If no *pid* is supplied, return the location of the standard configuration file. If *pid* is the string “*current*”, the path to the configuration file containing the configuration specific to the currently running process, i.e. the configuration obtained by overriding the values from the standard configuration file with the values explicitly supplied when the currently running process was invoked, is returned. If *pid* is the string “*” a list of all configuration belonging to currently running *egon-data* processes is returned. This can be used for error checking, because there should only ever be one such file.

set_numexpr_threads ()

Sets maximum threads used by NumExpr

Returns *None*

settings () → dict[str, dict[str, str]]

Return a nested dictionary containing the configuration settings.

It’s a nested dictionary because the top level has command names as keys and dictionaries as values where the second level dictionary has command line switches applicable to the command as keys and the supplied values as values.

So you would obtain the `--database-name` configuration setting used by the current invocation of of *egon-data* via

```
settings() ["egon-data"] ["--database-name"]
```

10.4 dataset_configuration

10.5 datasets

10.5.1 DSM_cts_ind

Currently, there are differences in the aggregated and individual DSM time series. These are caused by the truncation of the values at zero.

The sum of the individual time series is a more accurate value than the aggregated time series used so far and should replace it in the future. Since the deviations are relatively small, a tolerance is currently accepted in the sanity checks. See [#1120](<https://github.com/openego/eGon-data/issues/1120>) for updates.

class **DsmPotential** (*dependencies*)

Bases: *egon.data.datasets.Dataset*

class **EgonDemandregioSitesIndElectricityDsmTimeseries** (***kwargs*)

Bases: *sqlalchemy.ext.declarative.api.Base*

application

bus

e_max

e_min

industrial_sites_id


```

    p_max
    p_min
    p_set
    scn_name
    target = {'schema': 'demand', 'table': 'egon_demandregio_sites_ind_electricity_dsm_t

class EgonEtragoElectricityCtsDsmTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus
    e_max
    e_min
    p_max
    p_min
    p_set
    scn_name
    target = {'schema': 'demand', 'table': 'egon_etrago_electricity_cts_dsm_timeseries'}

class EgonOsmIndLoadCurvesIndividualDsmTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus
    e_max
    e_min
    osm_id
    p_max
    p_min
    p_set
    scn_name
    target = {'schema': 'demand', 'table': 'egon_osm_ind_load_curves_individual_dsm_time

class EgonSitesIndLoadCurvesIndividualDsmTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus
    e_max
    e_min
    p_max
    p_min
    p_set
    scn_name
    site_id
    target = {'schema': 'demand', 'table': 'egon_sites_ind_load_curves_individual_dsm_ti

```

aggregate_components (*df_dsm_buses, df_dsm_links, df_dsm_stores*)

calc_ind_site_timeseries (*scenario*)

calculate_potentials (*s_flex, s_util, s_inc, s_dec, delta_t, dsm*)

Calculate DSM-potential per bus using the methods by Heitkoetter et. al.: <https://doi.org/10.1016/j.adapen.2020.100001>

Parameters

s_flex: float Feasability factor to account for socio-technical restrictions

s_util: float Average annual utilisation rate

s_inc: float Shiftable share of installed capacity up to which load can be increased considering technical limitations

s_dec: float Shiftable share of installed capacity up to which load can be decreased considering technical limitations

delta_t: int Maximum shift duration in hours

dsm: DataFrame List of existing buses with DSM-potential including timeseries of loads

create_dsm_components (*con, p_max, p_min, e_max, e_min, dsm, export_aggregated=True*)
Create components representing DSM. Parameters

con : Connection to database

p_max: DataFrame Timeseries identifying maximum load increase

p_min: DataFrame Timeseries identifying maximum load decrease

e_max: DataFrame Timeseries identifying maximum energy amount to be preponed

e_min: DataFrame Timeseries identifying maximum energy amount to be postponed

dsm: DataFrame List of existing buses with DSM-potential including timeseries of loads

create_table (*df, table, engine=Engine(postgresql+psycopg2://egon:***@127.0.0.1:59734/egon-data)*)
Create table

cts_data_import (*cts_cool_vent_ac_share*)
Import CTS data necessary to identify DSM-potential.

cts_share: float Share of cooling, ventilation and AC in CTS demand

data_export (*dsm_buses, dsm_links, dsm_stores, carrier*)
Export new components to database.

Parameters

- **dsm_buses** (*DataFrame*) – Buses representing locations of DSM-potential
- **dsm_links** (*DataFrame*) – Links connecting DSM-buses and DSM-stores
- **dsm_stores** (*DataFrame*) – Stores representing DSM-potential
- **carrier** (*str*) – Remark to be filled in column ‘carrier’ identifying DSM-potential

delete_dsm_entries (*carrier*)
Deletes DSM-components from database if they already exist before creating new ones.

Parameters

carrier: str Remark in column ‘carrier’ identifying DSM-potential

div_list (*lst: list, div: float*)

dsm_cts_ind (*con=Engine(postgresql+psycopg2://egon:***@127.0.0.1:59734/egon-data),*

cts_cool_vent_ac_share=0.22, ind_vent_cool_share=0.039, ind_vent_share=0.017)

Execute methodology to create and implement components for DSM considering a) CTS per osm-area: combined potentials of cooling, ventilation and air

conditioning

b) Industry per osm-area: combined potentials of cooling and ventilation

c) Industrial Sites: potentials of ventilation in sites of

“Wirtschaftszweig” (WZ) 23

d) Industrial Sites: potentials of sites specified by subsectors

identified by Schmidt (<https://zenodo.org/record/3613767#.YTsgwVtCRhG>): Paper, Recycled Paper, Pulp, Cement

Modelled using the methods by Heitkoetter et. al.: <https://doi.org/10.1016/j.adapen.2020.100001>

Parameters

- **con** – Connection to database
- **cts_cool_vent_ac_share** (*float*) – Share of cooling, ventilation and AC in CTS demand
- **ind_vent_cool_share** (*float*) – Share of cooling and ventilation in industry demand
- **ind_vent_share** (*float*) – Share of ventilation in industry demand in sites of WZ 23

dsm_cts_ind_individual (*cts_cool_vent_ac_share=0.22, ind_vent_cool_share=0.039, ind_vent_share=0.017)*

Execute methodology to create and implement components for DSM considering a) CTS per osm-area: combined potentials of cooling, ventilation and air

conditioning

b) Industry per osm-area: combined potentials of cooling and ventilation

c) Industrial Sites: potentials of ventilation in sites of

“Wirtschaftszweig” (WZ) 23

d) Industrial Sites: potentials of sites specified by subsectors

identified by Schmidt (<https://zenodo.org/record/3613767#.YTsgwVtCRhG>): Paper, Recycled Paper, Pulp, Cement

Modelled using the methods by Heitkoetter et. al.: <https://doi.org/10.1016/j.adapen.2020.100001>

Parameters

- **cts_cool_vent_ac_share** (*float*) – Share of cooling, ventilation and AC in CTS demand
- **ind_vent_cool_share** (*float*) – Share of cooling and ventilation in industry demand

- **ind_vent_share** (*float*) – Share of ventilation in industry demand in sites of WZ 23

dsm_cts_ind_processing()

ind_osm_data_import (*ind_vent_cool_share*)

Import industry data per osm-area necessary to identify DSM-potential.

ind_share: float Share of considered application in industry demand

ind_osm_data_import_individual (*ind_vent_cool_share*)

Import industry data per osm-area necessary to identify DSM-potential.

ind_share: float Share of considered application in industry demand

ind_sites_data_import ()

Import industry sites data necessary to identify DSM-potential.

ind_sites_vent_data_import (*ind_vent_share*, *wz*)

Import industry sites necessary to identify DSM-potential.

ind_vent_share: float Share of considered application in industry demand

wz: int Wirtschaftszweig to be considered within industry sites

ind_sites_vent_data_import_individual (*ind_vent_share*, *wz*)

Import industry sites necessary to identify DSM-potential.

ind_vent_share: float Share of considered application in industry demand

wz: int Wirtschaftszweig to be considered within industry sites

relate_to_schmidt_sites (*dsm*)

10.5.2 calculate_dlr

Use the concept of dynamic line rating(DLR) to calculate temporal depending capacity for HV transmission lines. Inspired mainly on Planungsgrundsätze-2020 Available at: <https://www.transnetbw.de/files/pdf/netzentwicklung/netzplanungsgrundsätze/UENB_PIGrS_Juli2020.pdf>

class Calculate_dlr (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Calculate DLR and assign values to each line in the db

Parameters

- ***No parameters required**
- ***Dependencies*** –
 - *DataBundle*
 - *Osmtgmod*
 - *WeatherData*
 - *FixEhvSubnetworks*
- ***Resulting tables*** –
 - *grid.egon_etrage_line_timeseries* is filled

name = 'dlr'

```
version = '0.0.1'
```

DLR_Regions (*weather_info_path*, *regions_shape_path*)

Calculate DLR values for the given regions

Parameters

- **weather_info_path** (*str*, *mandatory*) – path of the weather data downloaded from ERA5
- **regions_shape_path** (*str*, *mandatory*) – path to the shape file with the shape of the regions to analyze

dlr ()

Calculate DLR and assign values to each line in the db

Parameters *No parameters required

10.5.3 ch4_prod

The central module containing code dealing with importing CH4 production data for eGon2035.

For eGon2035, the gas produced in Germany can be natural gas or biogas. The source productions are geolocalised potentials described as PyPSA generators. These generators are not extendable and their overall production over the year is limited directly in eTraGo by values from the Netzentwicklungsplan Gas 2020–2030 (36 TWh natural gas and 10 TWh biogas), also stored in the table *scenario.egon_scenario_parameters*.

class CH4Production (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Insert the CH4 productions into the database for eGon2035

Insert the CH4 productions into the database for eGon2035 by using the function *import_gas_generators()*.

Dependencies

- *GasAreaseGon2035*
- *GasNodesAndPipes*

Resulting tables

- *grid.egon_etrango_generator* is extended

name = 'CH4Production'

version = '0.0.7'

import_gas_generators (*scn_name*='eGon2035')

Inserts list of gas production units into the database

To insert the gas production units into the database, the following steps are followed:

- cleaning of the database table *grid.egon_etrango_generator* of the CH4 generators of the specific scenario (eGon2035),
- call of the functions *load_NG_generators()* and *load_biogas_generators()* that respectively return dataframes containing the natural- and bio-gas production units in Germany,
- attribution of the *bus_id* to which each generator is connected (call the function *assign_gas_bus_id* from *egon.data.db*),
- aggregation of the CH4 productions with same properties at the same bus. The properties that should be the same in order that different generators are aggregated are:

- scenario
- carrier
- marginal cost: this parameter differentiates the natural gas generators from the biogas generators,
- addition of the missing columns: `scn_name`, `carrier` and `generator_id`,
- insertion of the generators into the database.

Parameters `scn_name` (*str*) – Name of the scenario.

Returns *None*

load_NG_generators (*scn_name*)

Define the fossil CH₄ production units in Germany

This function reads from the SciGRID_gas dataset the fossil CH₄ production units in Germany, adjusts and returns them. Natural gas production reference: SciGRID_gas dataset (datasets/gas_data/data/IGGIELGN_Production.csv downloaded in [download_SciGRID_gas_data](#)). For more information on this data, refer to the [SciGRID_gas IGGIELGN documentation](#).

Parameters `scn_name` (*str*) – Name of the scenario.

Returns `CH4_generators_list` (*pandas.DataFrame*) – Dataframe containing the natural gas production units in Germany

load_biogas_generators (*scn_name*)

Define the biogas production units in Germany

This function downloads the Biogaspartner Einspeiseatlas into (datasets/gas_data/Biogaspartner_Einspeiseatlas_Deutschland_202 reads the biogas production units in Germany data, adjusts and returns them. For more information on this data refer to the [Einspeiseatlas website](#).

Parameters `scn_name` (*str*) – Name of the scenario

Returns `CH4_generators_list` (*pandas.DataFrame*) – Dataframe containing the biogas production units in Germany

10.5.4 ch4_storages

The central module containing all code dealing with importing gas stores

This module contains the functions to import the existing methane stores in Germany and inserting them into the database. They are modelled as PyPSA stores and are not extendable.

class CH4Storages (*dependencies*)

Bases: [egon.data.datasets.Dataset](#)

Inserts the gas stores in Germany

Inserts the non extendable gas stores in Germany into the database for the scenarios eGon2035 and eGon100RE using the function [insert_ch4_storages\(\)](#).

Dependencies

- [GasAreaseGon2035](#)
- [GasAreaseGon2035](#)
- [GasNodesAndPipes](#)

Resulting tables

- `grid.egon_etrage_store` is extended

name = 'CH4Storages'

version = '0.0.3'

import_ch4_grid_capacity(*scn_name*)

Defines the gas stores modelling the store capacity of the grid

Define dataframe containing the modelling of the grid storage capacity. The whole storage capacity of the grid (130000 MWh, estimation of the Bundesnetzagentur) is split uniformly between all the German gas nodes of the grid (without consideration of the capacities of the pipes). In eGon100RE, the storage capacity of the grid is split between H2 and CH4 stores, with the same share as the pipeline capacities (value calculated in the p-e-s run).

Parameters

- **scn_name** (*str*) – Name of the scenario
- **carrier** (*str*) – Name of the carrier

Returns *Gas_storages_list* – List of gas stores in Germany modelling the gas grid storage capacity

import_installed_ch4_storages(*scn_name*)

Defines list of CH4 stores from the SciGRID_gas data

This function reads from the SciGRID_gas dataset the existing CH4 cavern stores in Germany, adjusts and returns them. Caverns reference: SciGRID_gas dataset (datasets/gas_data/data/IGGIELGN_Storages.csv downloaded in [download_SciGRID_gas_data](#)). For more information on these data, refer to the [SciGRID_gas IGGIELGN documentation](#).

Parameters **scn_name** (*str*) – Name of the scenario

Returns *Gas_storages_list* – Dataframe containing the CH4 cavern store units in Germany

insert_ch4_storages()

Overall function to import non extendable gas stores in Germany

This function inserts the methane stores in Germany for the scenarios eGon2035 and eGon100RE by using the function [insert_ch4_stores\(\)](#) and has no return.

insert_ch4_stores(*scn_name*)

Inserts gas stores for specific scenario

Insert non extendable gas stores for specific scenario in Germany by executing the following steps:

- Clean the database.
- For CH4 stores, call the functions [import_installed_ch4_storages\(\)](#) to get the CH4 cavern stores and [import_ch4_grid_capacity\(\)](#) to get the CH4 stores modelling the storage capacity of the grid.
- Aggregate the stores attached to the same bus.
- Add the missing columns: store_id, scn_name, carrier, e_cyclic.
- Insert the stores into the database.

Parameters **scn_name** (*str*) – Name of the scenario.

Returns *None*

10.5.5 chp_etrango

The central module containing all code dealing with chp for eTraGo.

class ChpEtrango (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Collect data related to combined heat and power plants for the eTraGo tool

This dataset collects data for combined heat and power plants and puts it into a format that is needed for the transmission grid optimisation within the tool eTraGo. This data is then writing into the corresponding tables that are read by eTraGo.

Dependencies

- *HeatEtrango*
- *Chp*

Resulting tables

- *grid.egon_etrango_link* is extended
- *grid.egon_etrango_generator* is extended

name = 'ChpEtrango'

version = '0.0.6'

insert ()

Insert combined heat and power plants into eTraGo tables.

Gas CHP plants are modeled as links to the gas grid, biomass CHP plants (only in eGon2035) are modeled as generators

Returns *None*.

insert_egon100re ()

Insert combined heat and power plants into eTraGo tables for the eGon100RE scenario.

Returns *None*.

10.5.6 database

setup ()

Initialize the local database used for data processing.

10.5.7 electrical_neighbours

The central module containing all code dealing with electrical neighbours

class ElectricalNeighbours (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Add lines, loads, generation and storage for electrical neighbours

This dataset creates data for modelling the considered foreign countries and writes that data into the database tables that can be read by the eTraGo tool. Neighbouring countries are modelled in a lower spatial resolution, in general one node per country is considered. Defined load timeseries as well as generation and storage capacities are connected to these nodes. The nodes are connected by AC and DC transmission lines with the German grid and other neighbouring countries considering the grid topology from ENTSO-E.

Dependencies

- *Tyndp*
- *PypsaEurSec*

Resulting tables

- *grid.egon_etrango_bus* is extended
- *grid.egon_etrango_link* is extended
- *grid.egon_etrango_line* is extended
- *grid.egon_etrango_load* is extended
- *grid.egon_etrango_load_timeseries* is extended
- *grid.egon_etrango_storage* is extended
- *grid.egon_etrango_generator* is extended
- *grid.egon_etrango_generator_timeseries* is extended
- *grid.egon_etrango_transformer* is extended

name = 'ElectricalNeighbours'

version = '0.0.7'

buses (*scenario, sources, targets*)

Insert central buses in foreign countries per scenario

Parameters

- **sources** (*dict*) – List of dataset sources
- **targets** (*dict*) – List of dataset targets

Returns **central_buses** (*geopandas.GeoDataFrame*) – Buses in the center of foreign countries

calc_capacities ()

Calculates installed capacities from TYNDP data

Returns *pandas.DataFrame* – Installed capacities per foreign node and energy carrier

central_buses_egon100 (*sources*)

Returns buses in the middle of foreign countries based on eGon100RE

Parameters **sources** (*dict*) – List of sources

Returns *pandas.DataFrame* – Buses in the center of foreign countries

central_transformer (*scenario, sources, targets, central_buses, new_lines*)

Connect central foreign buses with different voltage levels

Parameters

- **sources** (*dict*) – List of dataset sources
- **targets** (*dict*) – List of dataset targets
- **central_buses** (*geopandas.GeoDataFrame*) – Buses in the center of foreign countries
- **new_lines** (*geopandas.GeoDataFrame*) – Lines that connect cross-border lines to central bus per country

Returns *None*.

choose_transformer (*s_nom*)

Select transformer and parameters from existing data in the grid model

It is assumed that transformers in the foreign countries are not limiting the electricity flow, so the capacity *s_nom* is set to the minimum sum of attached AC-lines. The electrical parameters are set according to already inserted transformers in the grid model for Germany.

Parameters *s_nom* (*float*) – Minimal sum of nominal power of lines at one side

Returns

- *int* – Selected transformer nominal power
- *float* – Selected transformer nominal impedance

cross_border_lines (*scenario, sources, targets, central_buses*)

Adds lines which connect border-crossing lines from osmtgmod to the central buses in the corresponding neighbouring country

Parameters

- **sources** (*dict*) – List of dataset sources
- **targets** (*dict*) – List of dataset targets
- **central_buses** (*geopandas.GeoDataFrame*) – Buses in the center of foreign countries

Returns *new_lines* (*geopandas.GeoDataFrame*) – Lines that connect cross-border lines to central bus per country

foreign_dc_lines (*scenario, sources, targets, central_buses*)

Insert DC lines to foreign countries manually

Parameters

- **sources** (*dict*) – List of dataset sources
- **targets** (*dict*) – List of dataset targets
- **central_buses** (*geopandas.GeoDataFrame*) – Buses in the center of foreign countries

Returns *None*.

get_cross_border_buses (*scenario, sources*)

Returns buses from osmTGmod which are outside of Germany.

Parameters *sources* (*dict*) – List of sources

Returns *geopandas.GeoDataFrame* – Electricity buses outside of Germany

get_cross_border_lines (*scenario, sources*)

Returns lines from osmTGmod which end or start outside of Germany.

Parameters *sources* (*dict*) – List of sources

Returns *geopandas.GeoDataFrame* – AC-lines outside of Germany

get_foreign_bus_id ()

Calculate the etrago bus id from Nodes of TYNDP based on the geometry

Returns *pandas.Series* – List of mapped node_ids from TYNDP and etragos bus_id

get_map_buses ()

Returns a dictionary of foreign regions which are aggregated to another

Returns *Combination of aggregated regions*

grid()

Insert electrical grid components for neighbouring countries

Returns *None*.**insert_generators** (*capacities*)

Insert generators for foreign countries based on TYNDP-data

Parameters **capacities** (*pandas.DataFrame*) – Installed capacities per foreign node and energy carrier**Returns** *None*.**insert_storage** (*capacities*)

Insert storage units for foreign countries based on TYNDP-data

Parameters **capacities** (*pandas.DataFrame*) – Installed capacities per foreign node and energy carrier**Returns** *None*.**map_carriers_tyndp** ()Map carriers from TYNDP-data to carriers used in eGon :returns: *dict* – Carrier from TYNDP and eGon**tyndp_demand** ()

Copy load timeseries data from TYNDP 2020. According to NEP 2021, the data for 2030 and 2040 is interpolated linearly.

Returns *None*.**tyndp_generation** ()

Insert data from TYNDP 2020 according to NEP 2021 Scenario ‘Distributed Energy’, linear interpolate between 2030 and 2040

Returns *None*.

10.5.8 electricity_demand_etrango

The central module containing code to merge data on electricity demand and feed this data into the corresponding eTraGo tables.

class ElectricalLoadEtrango (*dependencies*)Bases: *egon.data.datasets.Dataset***demands_per_bus** (*scenario*)

Sum all electricity demand curves up per bus

Parameters **scenario** (*str*) – Scenario name.**Returns** *pandas.DataFrame* – Aggregated electrical demand timeseries per bus**export_to_db** ()

Prepare and export eTraGo-ready information of loads per bus and their time series to the database

Returns *None*.**store_national_profiles** (*ind_curves_sites*, *ind_curves_osm*, *cts_curves*, *hh_curves*, *scenario*)

Store electrical load timeseries aggregated for national level as an input for pypsa-eur-sec

Parameters

- **ind_curves_sites** (*pd.DataFrame*) – Industrial load timeseries for industrial sites per bus

- **ind_curves_osm** (*pd.DataFrame*) – Industrial load timeseries for industrial osm areas per bus
- **cts_curves** (*pd.DataFrame*) – CTS load curves per bus
- **hh_curves** (*pd.DataFrame*) – Household load curves per bus
- **scenario** (*str*) – Scenario name

Returns *None*.

10.5.9 era5

Central module containing all code dealing with importing era5 weather data.

```
class EgonEra5Cells (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    geom
    geom_point
    w_id
```

```
class EgonRenewableFeedIn (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    carrier
    feedin
    w_id
    weather_year
```

```
class WeatherData (dependencies)
    Bases: egon.data.datasets.Dataset
```

Download weather data from ERA5 using atlite

This dataset downloads weather data for the selected representative weather year. This is done by applying functions from the atlite-tool. The downloaded weather data is stored into files within the subfolder 'cutouts'.

Dependencies

- *ScenarioParameters*
- *Vg250*
- *Setup*

Resulting tables

- *supply.egon_era5_weather_cells* is created and filled
- *supply.egon_era5_renewable_feedin* is created

```
name = 'Era5'
```

```
version = '0.0.2'
```

```
create_tables ()
```

```
download_era5 ()
```

Download weather data from era5

Returns *None*.

import_cutout (*boundary='Europe'*)

Import weather data from cutout

Returns **cutout** (*atlite.cutout.Cutout*) – Weather data stored in cutout

insert_weather_cells ()

Insert weather cells from era5 into database table

Returns *None*.

10.5.10 etrago_helpers

Module for repeated bus insertion tasks

copy_and_modify_buses (*from_scn, to_scn, filter_dict*)

Copy buses from one scenario to a different scenario

Parameters

- **from_scn** (*str*) – Source scenario.
- **to_scn** (*str*) – Target scenario.
- **filter_dict** (*dict*) – Filter buses according the information provided in this dict.

copy_and_modify_links (*from_scn, to_scn, carriers, sector*)

Copy links from one scenario to a different one.

Parameters

- **from_scn** (*str*) – Source scenario.
- **to_scn** (*str*) – Target scenario.
- **carriers** (*list*) – List of store carriers to copy.
- **sector** (*str*) – Name of sector (e.g. 'gas') to get cost information from.

copy_and_modify_stores (*from_scn, to_scn, carriers, sector*)

Copy stores from one scenario to a different one.

Parameters

- **from_scn** (*str*) – Source scenario.
- **to_scn** (*str*) – Target scenario.
- **carriers** (*list*) – List of store carriers to copy.
- **sector** (*str*) – Name of sector (e.g. 'gas') to get cost information from.

finalize_bus_insertion (*bus_data, carrier, target, scenario='eGon2035'*)

Finalize bus insertion to etrago table

Parameters

- **bus_data** (*geopandas.GeoDataFrame*) – GeoDataFrame containing the processed bus data.
- **carrier** (*str*) – Name of the carrier.
- **target** (*dict*) – Target schema and table information.
- **scenario** (*str, optional*) – Name of the scenario The default is 'eGon2035'.

Returns **bus_data** (*geopandas.GeoDataFrame*) – GeoDataFrame containing the inserted bus data.

```
initialise_bus_insertion (carrier, target, scenario='eGon2035')
```

Initialise bus insertion to etrago table

Parameters

- **carrier** (*str*) – Name of the carrier.
- **target** (*dict*) – Target schema and table information.
- **scenario** (*str, optional*) – Name of the scenario The default is 'eGon2035'.

Returns **gdf** (*geopandas.GeoDataFrame*) – Empty GeoDataFrame to store buses to.

10.5.11 etrago_setup

```
class EgonPfHvBus (**kwargs)
```

Bases: sqlalchemy.ext.declarative.api.Base

bus_id

carrier

country

geom

scn_name

type

v_mag_pu_max

v_mag_pu_min

v_mag_pu_set

v_nom

x

y

```
class EgonPfHvBusTimeseries (**kwargs)
```

Bases: sqlalchemy.ext.declarative.api.Base

bus_id

scn_name

v_mag_pu_set

```
class EgonPfHvBusmap (**kwargs)
```

Bases: sqlalchemy.ext.declarative.api.Base

bus0

bus1

path_length

scn_name

version

```
class EgonPfHvCarrier (**kwargs)
```

Bases: sqlalchemy.ext.declarative.api.Base

co2_emissions

```
color
commentary
name
nice_name
class EgonPfHvGenerator(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    build_year
    bus
    capital_cost
    carrier
    committable
    control
    down_time_before
    e_nom_max
    efficiency
    generator_id
    lifetime
    marginal_cost
    min_down_time
    min_up_time
    p_max_pu
    p_min_pu
    p_nom
    p_nom_extendable
    p_nom_max
    p_nom_min
    p_set
    q_set
    ramp_limit_down
    ramp_limit_shut_down
    ramp_limit_start_up
    ramp_limit_up
    scn_name
    shut_down_cost
    sign
    start_up_cost
```

```
type
up_time_before
class EgonPfHvGeneratorTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    generator_id
    marginal_cost
    p_max_pu
    p_min_pu
    p_set
    q_set
    scn_name
    temp_id
class EgonPfHvLine(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    b
    build_year
    bus0
    bus1
    cables
    capital_cost
    carrier
    g
    geom
    length
    lifetime
    line_id
    num_parallel
    r
    s_max_pu
    s_nom
    s_nom_extendable
    s_nom_max
    s_nom_min
    scn_name
    terrain_factor
    topo
    type
```



```
v_ang_max
v_ang_min
v_nom
x

class EgonPfHvLineTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    line_id
    s_max_pu
    scn_name
    temp_id

class EgonPfHvLink(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    build_year
    bus0
    bus1
    capital_cost
    carrier
    efficiency
    geom
    length
    lifetime
    link_id
    marginal_cost
    p_max_pu
    p_min_pu
    p_nom
    p_nom_extendable
    p_nom_max
    p_nom_min
    p_set
    scn_name
    terrain_factor
    topo
    type

class EgonPfHvLinkTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    efficiency
```

```
link_id
marginal_cost
p_max_pu
p_min_pu
p_set
scn_name
temp_id
class EgonPfHvLoad(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus
    carrier
    load_id
    p_set
    q_set
    scn_name
    sign
    type
class EgonPfHvLoadTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    load_id
    p_set
    q_set
    scn_name
    temp_id
class EgonPfHvStorage(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    build_year
    bus
    capital_cost
    carrier
    control
    cyclic_state_of_charge
    efficiency_dispatch
    efficiency_store
    inflow
    lifetime
    marginal_cost
```

```

    max_hours
    p_max_pu
    p_min_pu
    p_nom
    p_nom_extendable
    p_nom_max
    p_nom_min
    p_set
    q_set
    scn_name
    sign
    standing_loss
    state_of_charge_initial
    state_of_charge_set
    storage_id
    type
class EgonPfHvStorageTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    inflow
    marginal_cost
    p_max_pu
    p_min_pu
    p_set
    q_set
    scn_name
    state_of_charge_set
    storage_id
    temp_id
class EgonPfHvStore(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    build_year
    bus
    capital_cost
    carrier
    e_cyclic
    e_initial
    e_max_pu

```

```
e_min_pu
e_nom
e_nom_extendable
e_nom_max
e_nom_min
lifetime
marginal_cost
p_set
q_set
scn_name
sign
standing_loss
store_id
type
class EgonPfHvStoreTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    e_max_pu
    e_min_pu
    marginal_cost
    p_set
    q_set
    scn_name
    store_id
    temp_id
class EgonPfHvTempResolution(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    resolution
    start_time
    temp_id
    timesteps
class EgonPfHvTransformer(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    b
    build_year
    bus0
    bus1
    capital_cost
```

```

g
geom
lifetime
model
num_parallel
phase_shift
r
s_max_pu
s_nom
s_nom_extendable
s_nom_max
s_nom_min
scn_name
tap_position
tap_ratio
tap_side
topo
trafo_id
type
v_ang_max
v_ang_min
x

class EgonPfHvTransformerTimeseries(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    s_max_pu
    scn_name
    temp_id
    trafo_id

class EtragoSetup(dependencies)
    Bases: egon.data.datasets.Dataset

check_carriers()
    Check if any eTraGo table has carriers not included in the carrier table.

    Raises
        • ValueError if carriers that are not defined in the carriers table are
        • used in any eTraGo table.

create_tables()
    Create tables for eTraGo input data. :returns: None.

```

insert_carriers ()

Insert list of carriers into eTraGo table

Returns *None*.**link_geom_from_buses** (*df*, *scn_name*)

Add LineString geometry according to geometry of buses to links

Parameters

- **df** (*pandas.DataFrame*) – List of eTraGo links with bus0 and bus1 but without topology
- **scn_name** (*str*) – Scenario name

Returns **gdf** (*geopandas.GeoDataFrame*) – List of eTraGo links with bus0 and bus1 but with topology**temp_resolution** ()

Insert temporal resolution for etrago

Returns *None*.

10.5.12 fill_etrago_gen

class Egon_etrago_gen (*dependencies*)Bases: *egon.data.datasets.Dataset*

Group generators based on Scenario, carrier and bus. Marginal costs are assigned to generators without this data. Grouped generators are sent to the `egon_etrago_generator` table and a timeseries is assigned to the weather dependent ones.

Dependencies

- *PowerPlants*
- *WeatherData*

Resulting tables

- `:py:class:grid.egon_etrago_generator`

`<egon.data.datasets.etrago_setup.EgonPfHvGenerator>` is extended * *grid.egon_etrago_generator_timeseries* is filled

name = `'etrago_generators'`**version** = `'0.0.8'`**add_marginal_costs** (*power_plants*)**adjust_renew_feedin_table** (*renew_feedin*, *cfg*)**consistency** (*data*)**delete_previous_gen** (*cfg*, *con*, *etrago_gen_orig*, *power_plants*)**fill_etrago_gen_table** (*etrago_pp2*, *etrago_gen_orig*, *cfg*, *con*)**fill_etrago_gen_time_table** (*etrago_pp*, *power_plants*, *renew_feedin*, *pp_time*, *cfg*, *con*)**fill_etrago_generators** ()**group_power_plants** (*power_plants*, *renew_feedin*, *etrago_gen_orig*, *cfg*)**load_tables** (*con*, *cfg*)**numpy_nan** (*data*)

```
power_timeser(weather_data)
set_timeseries(power_plants, renew_feedin)
```

10.5.13 fix_ehv_subnetworks

The central module containing all code dealing with fixing ehv subnetworks

```
class FixEhvSubnetworks(dependencies)
    Bases: egon.data.datasets.Dataset
```

Manually fix grid topology in the extra high voltage grid to avoid subnetworks

This dataset includes fixes for the topology of the German extra high voltage grid. The initial grid topology from openstreetmap resp. osmTGmod includes some issues, eg. because of incomplete data. This dataset does not fix all those issues, but deals only with subnetworks in the extra high voltage grid that would result into problems in the grid optimisation.

Dependencies

- *Osmtgmod*

Resulting tables

- *grid.egon_etrango_bus* is updated
- *grid.egon_etrango_line* is updated
- *grid.egon_etrango_transformer* is updated

```
name = 'FixEhvSubnetworks'
```

```
version = '0.0.2'
```

```
add_bus(x, y, v_nom, scn_name)
add_line(x0, y0, x1, y1, v_nom, scn_name, cables)
add_trafo(x, y, v_nom0, v_nom1, scn_name, n=1)
drop_bus(x, y, v_nom, scn_name)
drop_line(x0, y0, x1, y1, v_nom, scn_name)
drop_trafo(x, y, v_nom0, v_nom1, scn_name)
fix_subnetworks(scn_name)
run()
select_bus_id(x, y, v_nom, scn_name, carrier)
```

10.5.14 gas_areas

The central module containing code to create CH4 and H2 voronoi polygons

```
class EgonPfHvGasVoronoi(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    Class definition of table grid.egon_gas_voronoi
    bus_id
        Bus of the corresponding area
```

carrier

Gas carrier of the voronoi area (“CH4”, “H2_grid” or “H2_saltcavern”)

geom

Geometry of the corresponding area

scn_name

Name of the scenario

class GasAreaseGon100RE (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Create the gas voronoi table and the gas voronoi areas for eGon100RE

Dependencies

- *EtragoSetup*
- *HydrogenBusEtrago*
- *HydrogenGridEtrago*
- *Vg250*
- *GasNodesAndPipes*
- *GasAreaseGon2035*

Resulting tables

- *EgonPfHvGasVoronoi*

name = 'GasAreaseGon100RE'

version = '0.0.1'

class GasAreaseGon2035 (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Create the gas voronoi table and the gas voronoi areas for eGon2035

Dependencies

- *EtragoSetup*
- *HydrogenBusEtrago*
- *Vg250*
- *GasNodesAndPipes*

Resulting tables

- *EgonPfHvGasVoronoi*

name = 'GasAreaseGon2035'

version = '0.0.2'

create_gas_voronoi_table ()

Create gas voronoi table

create_voronoi (*scn_name*, *carrier*)

Create voronoi polygons for specified carrier in specified scenario.

Parameters

- **scn_name** (*str*) – Name of the scenario

- **carrier** (*str*) – Name of the carrier

voronoi_egon100RE ()

Create voronoi polygons for all gas carriers in eGon100RE scenario

voronoi_egon2035 ()

Create voronoi polygons for all gas carriers in eGon2035 scenario

10.5.15 gas_grid

The module contains code used to insert the methane grid into the database

The central module contains all code dealing with the import of data from SciGRID_gas (IGGIELGN dataset) and inserting the CH4 buses and links into the database for the scenarios eGon2035 and eGon100RE.

The SciGRID_gas data downloaded with `download_SciGRID_gas_data()` into the folder `./datasets/gas_data/data` is also used by other modules.

In this module, only the IGGIELGN_Nodes and IGGIELGN_PipeSegments csv files are used in the function `insert_gas_data()` that inserts the CH4 buses and links, which for the case of gas represent pipelines, into the database.

class GasNodesAndPipes (*dependencies*)

Bases: `egon.data.datasets.Dataset`

Insert the CH4 buses and links into the database.

Insert the CH4 buses and links, which for the case of gas represent pipelines, into the database for the scenarios eGon2035 and eGon100RE with the functions `insert_gas_data()` and `insert_gas_data_eGon100RE()`.

Dependencies

- `DataBundle`
- `ElectricalNeighbours`
- `Osmtgmod`
- `ScenarioParameters`
- `EtragoSetup` (more specifically the `create_tables` task)

Resulting tables

- `grid.egon_etrango_bus` is extended
- `grid.egon_etrango_link` is extended

name = 'GasNodesAndPipes'

version = '0.0.9'

ch4_nodes_number_G (*gas_nodes_list*)

Return the number of CH4 buses in Germany

Parameters *gas_nodes_list* (*pandas.DataFrame*) – Dataframe containing the gas nodes in Europe

Returns *N_ch4_nodes_G* (*int*) – Number of CH4 buses in Germany

define_gas_buses_abroad (*scn_name*='eGon2035')

Define central CH4 buses in foreign countries for eGon2035

For the scenario eGon2035, define central CH₄ buses in foreign countries. The considered foreign countries are the direct neighbouring countries, with the addition of Russia that is considered as a source of fossil CH₄. Therefore, the following steps are executed:

- Definition of the foreign buses with the function `central_buses_egon100` from the module `electrical_neighbours`
- Removal of the superfluous buses in order to have only one bus in each neighbouring country
- Removal of the irrelevant columns
- Addition of the missing information: `scn_name` and `carrier`
- Attribution of an id to each bus

Parameters `scn_name` (*str*) – Name of the scenario

Returns `gdf_abroad_buses` (*pandas.DataFrame*) – Dataframe containing the gas buses in the neighbouring countries and one in the center of Germany in test mode

define_gas_nodes_list ()

Define list of CH₄ buses from SciGRID_gas IGGIELGN data

The CH₄ nodes are modelled as buses. Therefore the SciGRID_gas nodes are read from the IGGIELGN_Nodes csv file previously downloaded in the function `download_SciGRID_gas_data()`, corrected (erroneous country), and returned in a dataframe.

Returns `gas_nodes_list` (*pandas.DataFrame*) – Dataframe containing the gas nodes in Europe

define_gas_pipeline_list (*gas_nodes_list*, *abroad_gas_nodes_list*, *scn_name*='eGon2035')

Define gas pipelines in Germany from SciGRID_gas IGGIELGN data

The gas pipelines, modelled as PyPSA links are read from the IGGIELGN_PipeSegments csv file previously downloaded in the function `download_SciGRID_gas_data()`.

The capacities of the pipelines are determined by the correspondance table given by the parameters for the classification of gas pipelines in [Electricity, heat, and gas sector data for modeling the German system](#) related to the pipeline diameter given in the SciGRID_gas dataset.

The manual corrections allow to:

- Delete gas pipelines disconnected of the rest of the gas grid
- Connect one pipeline (also connected to Norway) disconnected of the rest of the gas grid
- Correct countries of some erroneous pipelines

Parameters

- `gas_nodes_list` (*dataframe*) – Dataframe containing the gas nodes in Europe
- `abroad_gas_nodes_list` (*dataframe*) – Dataframe containing the gas buses in the neighbouring countries and one in the center of Germany in test mode
- `scn_name` (*str*) – Name of the scenario

Returns `gas_pipelines_list` (*pandas.DataFrame*) – Dataframe containing the gas pipelines in Germany

download_SciGRID_gas_data ()

Download SciGRID_gas IGGIELGN data from Zenodo

The following data for CH₄ is downloaded into the folder `./datasets/gas_data/data`:

- Buses (file IGGIELGN_Nodes.csv),
- Pipelines (file IGGIELGN_PipeSegments.csv),
- Productions (file IGGIELGN_Productions.csv),
- Storages (file IGGIELGN_Storages.csv),
- LNG terminals (file IGGIELGN_LNGs.csv).

For more information on this data refer, to the [SciGRID_gas IGGIELGN documentation](#).

Returns *None*

insert_CH4_nodes_list (*gas_nodes_list*)

Insert list of German CH4 nodes into the database for eGon2035

Insert the list of German CH4 nodes into the database by executing the following steps:

- Receive the buses as parameter (from SciGRID_gas IGGIELGN data)
- Add the missing information: *scn_name* and *carrier*
- Clean the database table *grid.egon_etrago_bus* of the CH4 buses of the specific scenario (eGon2035) in Germany
- Insert the buses in the table *grid.egon_etrago_bus*

Parameters *gas_nodes_list* (*pandas.DataFrame*) – Dataframe containing the gas nodes in Europe

Returns *None*

insert_gas_buses_abroad (*scn_name='eGon2035'*)

Insert CH4 buses in neighbouring countries into database for eGon2035

- Definition of the CH4 buses abroad with the function *define_gas_buses_abroad()*
- Cleaning of the database table *grid.egon_etrago_bus* of the foreign CH4 buses of the specific scenario (eGon2035)
- Insertion of the neighbouring buses into the table *grid.egon_etrago_bus*.

Parameters *scn_name* (*str*) – Name of the scenario

Returns *gdf_abroad_buses* (*dataframe*) – Dataframe containing the CH4 buses in the neighbouring countries and one in the center of Germany in test mode

insert_gas_data ()

Function for importing methane data for eGon2035

This function imports the methane data (buses and pipelines) for eGon2035, by executing the following steps:

- Download the SciGRID_gas datasets with the function *download_SciGRID_gas_data()*
- Define CH4 buses with the function *define_gas_nodes_list()*
- Insert the CH4 buses in Germany into the database with the function *insert_CH4_nodes_list()*
- Insert the CH4 buses abroad into the database with the function *insert_gas_buses_abroad()*
- Insert the CH4 links representing the CH4 pipeline into the database with the function *insert_gas_pipeline_list()*
- Remove the isolated CH4 buses directly from the database using the function *remove_isolated_gas_buses()*

Returns *None*

insert_gas_data_eGon100RE ()

Function for importing methane data for eGon100RE

This function imports the methane data (buses and pipelines) for eGon100RE, by copying the CH₄ buses from the eGon2035 scenario using the function `copy_and_modify_buses` from the module `etrago_helpers`. The methane pipelines are also copied and their capacities are adapted: one share of the methane grid is retrofitted into an hydrogen grid, so the methane pipelines nominal capacities are reduced from this share (calculated in the pyspa-eur-sec run).

Returns *None*

insert_gas_pipeline_list (gas_pipelines_list, scn_name='eGon2035')

Insert list of gas pipelines into the database

Receive as argument a list of gas pipelines and insert them into the database after cleaning it.

Parameters

- **gas_pipelines_list** (*pandas.DataFrame*) – Dataframe containing the gas pipelines in Germany
- **scn_name** (*str*) – Name of the scenario

Returns *None*

remove_isolated_gas_buses ()

Delete CH₄ buses which are disconnected of the CH₄ grid for the eGon2035 scenario

Returns *None*

10.5.16 generate_voronoi

The central module containing code to create CH₄ and H₂ voronoi polygons

get_voronoi_geodataframe (buses, boundary)

Create voronoi polygons for the passed buses within the boundaries.

Parameters

- **buses** (*geopandas.GeoDataFrame*) – Buses to create the voronoi for.
- **boundary** (*Multipolygon, Polygon*) – Bounding box for the voronoi generation.

Returns **gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing the bus_ids and the respective voronoi polygons.

10.5.17 heat_demand_europe

Central module containing all code downloading hotmaps heat demand data.

The 2050 national heat demand of the Hotmaps current policy scenario for buildings are used in the eGon100RE scenario for assumptions on national heating demands in European countries, but not for Germany. The data are downloaded to be used in the PyPSA-Eur-Sec scenario generator (forked into open_ego).

class HeatDemandEurope (*dependencies*)

Bases: `egon.data.datasets.Dataset`

Downloads annual heat demands for European countries from hotmaps

This dataset downloads annual heat demands for all European countries for the year 2050 from hotmaps and stores the results into files. These are later used by pypsa-eur-sec.

Dependencies

- *Setup*

name = 'heat-demands-europe'

version = 'scen_current_building_demand.csv_hotmaps.0.1'

download()

Download Hotmaps current policy scenario for building heat demands.

The downloaded data contain residential and non-residential-sector national heat demands for different years.

Parameters None

Returns None

10.5.18 industrial_gas_demand

The central module containing code dealing with gas industrial demand

In this module, the functions to import the industrial hydrogen and methane demands from the opendata.ffe database and to insert them into the database after modification are to be found.

class IndustrialGasDemand (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Download the industrial gas demands from the opendata.ffe database

Data is downloaded to the folder `./datasets/gas_data/demand` using the function `download_industrial_gas_demand()` and no dataset is resulting.

Dependencies

- *ScenarioParameters*

name = 'IndustrialGasDemand'

version = '0.0.4'

class IndustrialGasDemandeGon100RE (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Insert the hourly resolved industrial gas demands into the database for eGon100RE

Insert the industrial methane and hydrogen demands and their associated time series for the scenario eGon100RE by executing the function `insert_industrial_gas_demand_egon100RE()`.

Dependencies

- *GasAreaseGon100RE*
- *GasNodesAndPipes*
- *HydrogenBusEtrago*
- *IndustrialGasDemand*

Resulting tables

- `grid.egon_etrargo_load` is extended
- `grid.egon_etrargo_load_timeseries` is extended

```
name = 'IndustrialGasDemandeGon100RE'
```

```
version = '0.0.3'
```

```
class IndustrialGasDemandeGon2035 (dependencies)
```

Bases: `egon.data.datasets.Dataset`

Insert the hourly resolved industrial gas demands into the database for eGon2035

Insert the industrial methane and hydrogen demands and their associated time series for the scenario eGon2035 by executing the function `insert_industrial_gas_demand_egon2035()`.

Dependencies

- `GasAreaseGon2035`
- `GasNodesAndPipes`
- `HydrogenBusEtrago`
- `IndustrialGasDemand`

Resulting tables

- `grid.egon_etrago_load` is extended
- `grid.egon_etrago_load_timeseries` is extended

```
name = 'IndustrialGasDemandeGon2035'
```

```
version = '0.0.3'
```

```
delete_old_entries (scn_name)
```

Delete CH₄ and H₂ loads and load time series for the specified scenario

Parameters `scn_name` (*str*) – Name of the scenario.

Returns *None*

```
download_industrial_gas_demand ()
```

Download the industrial gas demand data from opendata.ffe database

The industrial demands for hydrogen and methane are downloaded in the folder `./datasets/gas_data/demand`. These loads are hourly and NUTS3-level resolved. For more information on these data, refer to the [Extremos project documentation](#).

Returns *None*

```
insert_industrial_gas_demand_egon100RE ()
```

Insert industrial gas demands into the database for eGon100RE

Insert the industrial CH₄ and H₂ demands and their associated time series into the database for the eGon100RE scenario. The data, previously downloaded in `download_industrial_gas_demand()` are adapted by executing the following steps:

- Clean the database with the function `delete_old_entries()`
- Read and prepare the CH₄ and the H₂ industrial demands and their associated time series in Germany with the function `read_and_process_demand()`
- Identify and adjust the total industrial CH₄ and H₂ loads for Germany generated by PyPSA-Eur-Sec
 - For CH₄, the time series used is the one from H₂, because the industrial CH₄ demand in the opendata.ffe database is 0

- In test mode, the total values are obtained by evaluating the share of H2 demand in the test region (NUTS1: DEF, Schleswig-Holstein) with respect to the H2 demand in full Germany model (NUTS0: DE). This task has been outsourced to save processing cost.
- Aggregate the demands with the same properties at the same gas bus
- Insert the loads into the database by executing `insert_new_entries()`
- Insert the time series associated to the loads into the database by executing `insert_industrial_gas_demand_time_series()`

Returns *None*

insert_industrial_gas_demand_egon2035()

Insert industrial gas demands into the database for eGon2035

Insert the industrial CH4 and H2 demands and their associated time series into the database for the eGon2035 scenario. The data previously downloaded in `download_industrial_gas_demand()` is adjusted by executing the following steps:

- Clean the database with the function `delete_old_entries()`
- Read and prepare the CH4 and the H2 industrial demands and their associated time series in Germany with the function `read_and_process_demand()`
- Aggregate the demands with the same properties at the same gas bus
- Insert the loads into the database by executing `insert_new_entries()`
- Insert the time series associated to the loads into the database by executing `insert_industrial_gas_demand_time_series()`

Returns *None*

insert_industrial_gas_demand_time_series(egon_etrago_load_gas)

Insert list of industrial gas demand time series (one per NUTS3 region)

These loads are hourly and on NUTS3 level resolved.

Parameters **industrial_gas_demand** (*pandas.DataFrame*) – Dataframe containing the loads that have been inserted into the database and whose time series will be inserted into the database.

Returns *None*

insert_new_entries(industrial_gas_demand, scn_name)

Insert industrial gas loads into the database

This function prepares and imports the industrial gas loads by executing the following steps:

- Attribution of an id to each load in the list received as parameter
- Deletion of the column containing the time series (they will be inserted in another table (grid.egon_etrago_load_timeseries) in the `insert_industrial_gas_demand_time_series()`)
- Insertion of the loads into the database
- Return of the dataframe still containing the time series columns

Parameters

- **industrial_gas_demand** (*pandas.DataFrame*) – Load data to insert (containing the time series)

- **scn_name** (*str*) – Name of the scenario.

Returns **industrial_gas_demand** (*pandas.DataFrame*) – Dataframe containing the loads that have been inserted in the database with their time series

read_and_process_demand (*scn_name='eGon2035', carrier=None, grid_carrier=None*)

Assign the industrial gas demand in Germany to buses

This function prepares and returns the industrial gas demand time series for CH4 or H2 and for a specific scenario by executing the following steps:

- Read the industrial demand time series in Germany with the function `read_industrial_demand()`
- Attribute the bus_id to which each load and its associated time series is associated by calling the function `assign_gas_bus_id` from `egon.data.db`
- Adjust the columns: add “carrier” and remove useless ones

Parameters

- **scn_name** (*str*) – Name of the scenario
- **carrier** (*str*) – Name of the carrier, the demand should hold
- **grid_carrier** (*str*) – Carrier name of the buses, the demand should be assigned to

Returns **industrial_demand** (*pandas.DataFrame*) – Dataframe containing the industrial demand in Germany

read_industrial_demand (*scn_name, carrier*)

Read the industrial gas demand data in Germany

This function reads the methane or hydrogen industrial demand time series previously downloaded in `download_industrial_gas_demand()` for the scenarios eGon2035 or eGon100RE.

Parameters

- **scn_name** (*str*) – Name of the scenario
- **carrier** (*str*) – Name of the gas carrier

Returns **df** (*pandas.DataFrame*) – Dataframe containing the industrial gas demand time series

10.5.19 mastr

Download Marktstammdatenregister (MaStR) datasets unit registry. It incorporates two different datasets:

Dump 2021-05-03 * Source: <https://sandbox.zenodo.org/record/808086> * Used technologies: PV plants, wind turbines, biomass, hydro plants,

combustion, nuclear, gsgk, storage

- Data is further processed in dataset `egon.data.datasets.power_plants.PowerPlants`

Dump 2022-11-17 * Source: <https://sandbox.zenodo.org/record/1132839> * Used technologies: PV plants, wind turbines, biomass, hydro plants * Data is further processed in module

`egon.data.datasets.power_plants.mastr PowerPlants`

Todo: Finish docstring TBD

download_mastr_data ()

Download MaStR data from Zenodo

10.5.20 mv_grid_districts

Medium-voltage grid districts describe the area supplied by one MV grid

Medium-voltage grid districts are defined by one polygon that represents the supply area. Each MV grid district is connected to the HV grid via a single substation.

The methods used for identifying the MV grid districts are heavily inspired by [Hülk et al. \(2017\)](#) (section 2.3), but the implementation differs in detail. The main difference is that direct adjacency is preferred over proximity. For polygons of municipalities without a substation inside, it is iteratively checked for direct adjacent other polygons that have a substation inside. Speaking visually, a MV grid district grows around a polygon with a substation inside.

The grid districts are identified using three data sources

1. Polygons of municipalities (`Vg250GemClean`)
2. HV-MV substations (`EgonHvmvSubstation`)
3. HV-MV substation voronoi polygons (`EgonHvmvSubstationVoronoi`)

Fundamentally, it is assumed that grid districts (supply areas) often go along borders of administrative units, in particular along the borders of municipalities due to the concession levy. Furthermore, it is assumed that one grid district is supplied via a single substation and that locations of substations and grid districts are designed for aiming least lengths of grid line and cables.

With these assumptions, the three data sources from above are processed as follows:

- Find the number of substations inside each municipality
- Split municipalities with more than one substation inside * Cut polygons of municipalities with voronoi polygons of respective substations
 - Assign resulting municipality polygon fragments to nearest substation
- Assign municipalities without a single substation to nearest substation in the neighborhood
- Merge all municipality polygons and parts of municipality polygons to a single polygon grouped by the assigned substation

For finding the nearest substation, as already said, direct adjacency is preferred over closest distance. This means, the nearest substation does not necessarily have to be the closest substation in the sense of beeline distance. But it is the substation definitely located in a neighboring polygon. This prevents the algorithm to find solutions where a MV grid districts consists of multi-polygons with some space in between. Nevertheless, beeline distance still plays an important role, as the algorithm acts in two steps

1. Iteratively look for neighboring polygons until there are no further polygons
2. Find a polygon to assign to by minimum beeline distance

The second step is required in order to cover edge cases, such as islands.

For understanding how this is implemented into separate functions, please see `define_mv_grid_districts()`.

```
class HvmvSubstPerMunicipality (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    ags_0
    area_ha
    bem
    bez
```

```
count_hole
gen
geometry
id
is_hole
nuts
old_id
path
rs_0
subst_count

class MvGridDistricts(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    area
    bus_id
    geom

class MvGridDistrictsDissolved(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    area
    bus_id
    geom
    id

class Vg250GemClean(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    ags_0
    area_ha
    bem
    bez
    count_hole
    gen
    geometry
    id
    is_hole
    nuts
    old_id
    path
    rs_0
```

```

class VoronoiMunicipalityCuts(**kwargs)
    Bases: egon.data.datasets.mv_grid_districts.VoronoiMunicipalityCutsBase,
           sqlalchemy.ext.declarative.api.Base

    ags_0
    bus_id
    geom
    geom_sub
    id
    municipality_id
    subst_count
    voronoi_id

class VoronoiMunicipalityCutsAssigned(**kwargs)
    Bases: egon.data.datasets.mv_grid_districts.VoronoiMunicipalityCutsBase,
           sqlalchemy.ext.declarative.api.Base

    ags_0
    bus_id
    geom
    geom_sub
    id
    municipality_id
    subst_count
    temp_id
    voronoi_id

class VoronoiMunicipalityCutsBase
    Bases: object

    ags_0 = Column(None, String(), table=None)
    bus_id = Column(None, Integer(), table=None)
    geom = Column(None, Geometry(geometry_type='POLYGON', srid=3035), table=None)
    geom_sub = Column(None, Geometry(geometry_type='POINT', srid=3035), table=None)
    municipality_id = Column(None, Integer(), table=None)
    subst_count = Column(None, Integer(), table=None)
    voronoi_id = Column(None, Integer(), table=None)

assign_substation_municipality_fragments(with_substation, without_substation, strategy,
                                         session)
    Assign bus_id from next neighboring polygon to municipality fragment

    For parts municipalities without a substation inside their polygon the next municipality polygon part is found
    and assigned.

    Resulting data including information about the assigned substation is saved to
    VoronoiMunicipalityCutsAssigned.

```

Parameters

- **with_substation** (*SQLAlchemy subquery*) – Polygons that have a substation inside or are assigned to a substation
- **without_substation** (*SQLAlchemy subquery*) – Subquery that includes polygons without a substation
- **strategy** (*str*) – Either
 - “touches”: Only polygons that touch another polygon from *with_substation* are considered
 - “within”: Only polygons within a radius of 100 km of polygons without substation are considered for assignment
- **session** (*SQLAlchemy session*) – SQLAlchemy session object

See also:

The (), but (), different ()

define_mv_grid_districts()

Define spatial extent of MV grid districts

The process of identifying the boundary of medium-voltage grid districts is organized in three steps

1. *substations_in_municipalities()*: The number of substations located inside each municipality is calculated
2. *split_multi_substation_municipalities()*: The municipalities with >1 substation inside are split by Voronoi polygons around substations
3. *merge_polygons_to_grid_district()*: All polygons are merged such that one polygon has exactly one single substation inside

Finally, intermediate tables used for storing data temporarily are deleted.

merge_polygons_to_grid_district()

Merge municipality polygon (parts) to MV grid districts

Polygons of municipalities and cut parts of such polygons are merged to a single grid district per one HV-MV substation. Prior determined assignment of cut polygons parts is used as well as proximity of entire municipality polygons to polygons with a substation inside.

- Step 1: Merge municipality parts that are assigned to the same substation
- Step 2: Insert municipality polygons with exactly one substation
- Step 3: Assign municipality polygons without a substation and insert to table
- Step 4: Merge MV grid district parts

nearest_polygon_with_substation (*with_substation, without_substation, strategy, session*)

Assign next neighboring polygon

For municipalities without a substation inside their polygon the next MV grid district (part) polygon is found and assigned.

Resulting data including information about the assigned substation is saved to *MvGridDistrictsDissolved*.

Parameters

- **with_substation** (*SQLAlchemy subquery*) – Polygons that have a substation inside or are assigned to a substation
- **without_substation** (*SQLAlchemy subquery*) – Subquery that includes polygons without a substation
- **strategy** (*str*) – Either
 - “touches”: Only polygons that touch another polygon from *with_substation* are considered
 - “within”: Only polygons within a radius of 100 km of polygons without substation are considered for assignment
- **session** (*SQLAlchemy session*) – SQLAlchemy session object

Returns *list* – IDs of polygons that were already assigned to a polygon with a substation

split_multi_substation_municipalities()

Split municipalities that have more than one substation

Municipalities that contain more than one HV-MV substation in their polygon are cut by HV-MV voronoi polygons. Resulting fragments are then assigned to the next neighboring polygon that has a substation.

In detail, the following steps are performed:

- Step 1: cut municipalities with voronoi polygons
- Step 2: Determine number of substations inside cut polygons
- Step 3: separate cut polygons with exactly one substation inside
- Step 4: Assign polygon without a substation to next neighboring polygon with a substation
- Step 5: Assign remaining polygons that are non-touching

substations_in_municipalities()

Create a table that counts number of HV-MV substations in each MV grid

Counting is performed in two steps

1. HV-MV substations are spatially joined on municipalities, grouped by municipality and number of substations counted
2. Because (1) works only for number of substations >0, all municipalities not containing a substation, are added

10.5.21 renewable_feedin

Central module containing all code dealing with processing era5 weather data.

```
class MapZensusWeatherCell (**kwargs)
```

```
    Bases: sqlalchemy.ext.declarative.api.Base
```

```
    w_id
```

```
    zensus_population_id
```

```
class RenewableFeedin (dependencies)
```

```
    Bases: egon.data.datasets.Dataset
```

```
    Calculate possible feedin time series for renewable energy generators
```

This dataset calculates possible feedin timeseries for fluctuation renewable generators and coefficient of performance time series for heat pumps. Relevant input is the downloaded weather data. Parameters for the time series calculation are also defined by representative types of pv plants and wind turbines that are selected within this dataset. The resulting profiles are stored in the database.

Dependencies

- *WeatherData*
- *Vg250*
- *ZensusVg250*

Resulting tables

- *supply.egon_era5_renewable_feedin* is filled

name = 'RenewableFeedin'

version = '0.0.7'

federal_states_per_weather_cell()

Assigns a federal state to each weather cell in Germany.

Sets the federal state to the weather cells using the centroid. Weather cells at the borders whose centroid is not inside Germany are assigned to the closest federal state.

Returns *GeoPandas.GeoDataFrame* – Index, points and federal state of weather cells inside Germany

feedin_per_turbine()

Calculate feedin timeseries per turbine type and weather cell

Returns *gdf (GeoPandas.GeoDataFrame)* – Feed-in timeseries per turbine type and weather cell

heat_pump_cop()

Calculate coefficient of performance for heat pumps according to T. Brown et al: “Synergies of sector coupling and transmission reinforcement in a cost-optimised, highly renewable European energy system”, 2018, p. 8

Returns *None*.

insert_feedin(data, carrier, weather_year)

Insert feedin data into database

Parameters

- **data** (*xarray.core.dataarray.DataArray*) – Feedin timeseries data
- **carrier** (*str*) – Name of energy carrier
- **weather_year** (*int*) – Selected weather year

Returns *None*.

mapping_zensus_weather()

Perform mapping between era5 weather cell and zensus grid

offshore_weather_cells(geom_column='geom')

Get weather cells which intersect with Germany

Returns *GeoPandas.GeoDataFrame* – Index and points of weather cells inside Germany

pv()

Insert feed-in timeseries for pv plants to database

Returns *None*.

solar_thermal()

Insert feed-in timeseries for pv plants to database

Returns *None*.

turbine_per_weather_cell()

Assign wind onshore turbine types to weather cells

Returns *weather_cells* (*GeoPandas.GeoDataFrame*) – Weather cells in Germany including turbine type

weather_cells_in_germany (*geom_column='geom'*)

Get weather cells which intersect with Germany

Returns *GeoPandas.GeoDataFrame* – Index and points of weather cells inside Germany

wind()

Insert feed-in timeseries for wind onshore turbines to database

Returns *None*.

wind_offshore()

Insert feed-in timeseries for wind offshore turbines to database

Returns *None*.

10.5.22 sanity_checks

This module does sanity checks for both the eGon2035 and the eGon100RE scenario separately where a percentage error is given to showcase difference in output and input values. Please note that there are missing input technologies in the supply tables. Authors: @ALonso, @dana, @nailend, @nesnoj, @khelfen

class SanityChecks (*dependencies*)

Bases: *egon.data.datasets.Dataset*

name = 'SanityChecks'

version = '0.0.8'

cts_electricity_demand_share (*rtol=1e-05*)

Sanity check for dataset electricity_demand_timeseries : CtsBuildings

Check sum of aggregated cts electricity demand share which equals to one for every substation as the substation profile is linearly disaggregated to all buildings.

cts_heat_demand_share (*rtol=1e-05*)

Sanity check for dataset electricity_demand_timeseries : CtsBuildings

Check sum of aggregated cts heat demand share which equals to one for every substation as the substation profile is linearly disaggregated to all buildings.

etrago_eGon2035_electricity ()

Execute basic sanity checks.

Returns print statements as sanity checks for the electricity sector in the eGon2035 scenario.

Parameters *None*

Returns *None*

etrago_eGon2035_gas_DE ()

Execute basic sanity checks for the gas sector in eGon2035

Returns print statements as sanity checks for the gas sector in the eGon2035 scenario for the following components in Germany:

- Buses: with the function `sanity_check_gas_buses()`
- Loads: for the carriers 'CH4_for_industry' and 'H2_for_industry' the deviation is calculated between the sum of the loads in the database and the sum the loads in the sources document (opendata.ffe database)
- Generators: the deviation is calculated between the sums of the nominal powers of the gas generators in the database and of the ones in the sources document (Biogaspartner Einspeiseatlas Deutschland from the dena and Productions from the SciGRID_gas data)
- Stores: deviations for stores with following carriers are calculated:
 - 'CH4': with the function `sanity_check_CH4_stores()`
 - 'H2_underground': with the function `sanity_check_H2_saltcavern_stores()`
- One-port components (loads, generators, stores): verification that they are all connected to a bus present in the data base with the function `sanity_check_gas_one_port()`
- **Links: verification:**
 - that the gas links are all connected to buses present in the data base with the function `sanity_check_gas_links()`
 - of the capacity of the gas grid with the function `sanity_check_CH4_grid()`

etrago_eGon2035_gas_abroad()

Execute basic sanity checks for the gas sector in eGon2035 abroad

Returns print statements as sanity checks for the gas sector in the eGon2035 scenario for the following components in Germany:

- Buses
- Loads: for the carriers 'CH4' and 'H2_for_industry' the deviation is calculated between the sum of the loads in the database and the sum in the sources document (TYNDP)
- Generators: the deviation is calculated between the sums of the nominal powers of the methane generators abroad in the database and of the ones in the sources document (TYNDP)
- Stores: the deviation for methane stores abroad is calculated between the sum of the capacities in the data base and the one of the source document (SciGRID_gas data)
- Links: verification of the capacity of the crossbordering gas grid pipelines.

etrago_eGon2035_heat()

Execute basic sanity checks.

Returns print statements as sanity checks for the heat sector in the eGon2035 scenario.

Parameters None

Returns None

residential_electricity_annual_sum(rtol=1e-05)

Sanity check for dataset electricity_demand_timeseries : Demand_Building_Assignment

Aggregate the annual demand of all census cells at NUTS3 to compare with initial scaling parameters from DemandRegio.

residential_electricity_hh_refinement(rtol=1e-05)

Sanity check for dataset electricity_demand_timeseries : Household Demands

Check sum of aggregated household types after refinement method was applied and compare it to the original census values.

sanity_check_CH4_grid (*scn*)

Execute sanity checks for the gas grid capacity in Germany

Returns print statements as sanity checks for the CH4 links (pipelines) in Germany. The deviation is calculated between the sum of the power (*p_nom*) of all the CH4 pipelines in Germany for one scenario in the database and the sum of the powers of the imported pipelines. In eGon100RE, the sum is reduced by the share of the grid that is allocated to hydrogen (share calculated by PyPSA-eur-sec). This test works also in test mode.

Parameters *scn_name* (*str*) – Name of the scenario

Returns *scn_name* (*float*) – Sum of the power (*p_nom*) of all the pipelines in Germany

sanity_check_CH4_stores (*scn*)

Execute sanity checks for the CH4 stores in Germany

Returns print statements as sanity checks for the CH4 stores capacity in Germany. The deviation is calculated between:

- the sum of the capacities of the stores with carrier ‘CH4’ in the database (for one scenario) and
- **the sum of:**
 - the capacity the gas grid allocated to CH4 (total capacity in eGon2035 and capacity reduced the share of the grid allocated to H2 in eGon100RE)
 - the total capacity of the CH4 stores in Germany (source: GIE)

Parameters *scn_name* (*str*) – Name of the scenario

sanity_check_H2_saltcavern_stores (*scn*)

Execute sanity checks for the H2 saltcavern stores in Germany

Returns print as sanity checks for the H2 saltcavern potential storage capacity in Germany. The deviation is calculated between:

- the sum of the of the H2 saltcavern potential storage capacity (*e_nom_max*) in the database and
- the sum of the H2 saltcavern potential storage capacity assumed to be the ratio of the areas of 500 m radius around substations in each german federal state and the estimated total hydrogen storage potential of the corresponding federal state (data from InSpEE-DS report).

This test works also in test mode.

Parameters *scn_name* (*str*) – Name of the scenario

sanity_check_gas_buses (*scn*)

Execute sanity checks for the gas buses in Germany

Returns print statements as sanity checks for the CH4, H2_grid and H2_saltcavern buses.

- For all of them, it is checked if they are not isolated.
- For the grid buses, the deviation is calculated between the number of gas grid buses in the database and the original Scigrd_gas number of gas buses in Germany.

Parameters *scn_name* (*str*) – Name of the scenario

sanity_check_gas_links (*scn*)

Check connections of gas links

Verify that gas links are all connected to buses present in the data base. Return print statements if this is not the case. This sanity check is not specific to Germany, it also includes the neighbouring countries.

Parameters `scn_name` (*str*) – Name of the scenario

sanity_check_gas_one_port (*scn*)

Check connections of gas one-port components

Verify that gas one-port component (loads, generators, stores) are all connected to a bus (of the right carrier) present in the data base. Return print statements if this is not the case. These sanity checks are not specific to Germany, they also include the neighbouring countries.

Parameters `scn_name` (*str*) – Name of the scenario

sanitycheck_dsm ()

sanitycheck_emobility_mit ()

Execute sanity checks for eMobility: motorized individual travel

Checks data integrity for eGon2035, eGon2035_lowflex and eGon100RE scenario using assertions:

1. Allocated EV numbers and EVs allocated to grid districts
2. Trip data (original inout data from simBEV)
3. Model data in eTraGo PF tables (grid.egon_etrigo_*)

Parameters `None`

Returns *None*

sanitycheck_home_batteries ()

sanitycheck_pv_rooftop_buildings ()

10.5.23 scenario_capacities

The central module containing all code dealing with importing data from Netzentwicklungsplan 2035, Version 2031, Szenario C

```
class EgonScenarioCapacities (**kwargs)
```

```
    Bases: sqlalchemy.ext.declarative.api.Base
```

```
    capacity
```

```
    carrier
```

```
    component
```

```
    index
```

```
    nuts
```

```
    scenario_name
```

```
class NEP2021ConvPowerPlants (**kwargs)
```

```
    Bases: sqlalchemy.ext.declarative.api.Base
```

```
    a2035_capacity
```

```
    a2035_chp
```

```
    b2035_capacity
```

```
    b2035_chp
```

b2040_capacity
b2040_chp
bnetza_id
c2035_capacity
c2035_chp
capacity
carrier
carrier_nep
chp
city
commissioned
federal_state
index
name
name_unit
postcode
status

class ScenarioCapacities (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Create and fill table with installed generation capacities in Germany

This dataset creates and fills a table with the installed generation capacities in Germany in a lower spatial resolution (either per federal state or on national level). This data is coming from external sources (e.g. German grid development plan for scenario eGon2035). The table is in downstream datasets used to define target values for the installed capacities.

Dependencies

- *Setup*
- *PypsaEurSec*
- *Vg250*
- *DataBundle*
- *ZensusPopulation*

Resulting tables

- *supply.egon_scenario_capacities* is created and filled
- *supply.egon_nep_2021_conventional_powerplants* is created and filled

name = 'ScenarioCapacities'

version = '0.0.13'

aggr_nep_capacities (*carriers*)

Aggregates capacities from NEP power plants list by carrier and federal state

Returns *pandas.DataFrame* – Dataframe with capacities per federal state and carrier

create_table()

Create input tables for scenario setup

Returns *None*.**district_heating_input()**

Imports data for district heating networks in Germany

Returns *None*.**eGon100_capacities()**

Inserts installed capacities for the eGon100 scenario

Returns *None*.**insert_capacities_per_federal_state_nep()**

Inserts installed capacities per federal state according to NEP 2035 (version 2021), scenario 2035 C

Returns *None*.**insert_data_nep()**

Overall function for importing scenario input data for eGon2035 scenario

Returns *None*.**insert_nep_list_powerplants** (*export=True*)

Insert list of conventional powerplants attached to the approval of the scenario report by BNetzA

Parameters **export** (*bool*) – Choose if nep list should be exported to the data base. The default is True. If export=False a data frame will be returned**Returns** **kw_liste_nep** (*pandas.DataFrame*) – List of conventional power plants from nep if export=False**map_carrier()**

Map carriers from NEP and Marktstammdatenregister to carriers from eGon

Returns *pandas.Series* – List of mapped carriers**nuts_mapping()****population_share()**

Calculate share of population in testmode

Returns *float* – Share of population in testmode

10.5.24 society_prognosis

The central module containing all code dealing with processing and forecast Zensus data.

class EgonHouseholdPrognosis (***kwargs*)Bases: *sqlalchemy.ext.declarative.api.Base***households****year****zensus_population_id****class EgonPopulationPrognosis** (***kwargs*)Bases: *sqlalchemy.ext.declarative.api.Base***population****year**

```

    zensus_population_id
class SocietyPrognosis (dependencies)
    Bases: egon.data.datasets.Dataset
create_tables ()
    Create table to map zensus grid and administrative districts (nuts3)
household_prognosis_per_year (prognosis_nuts3, zensus, year)
    Calculate household prognosis for a specific year
zensus_household ()
    Bring household prognosis from DemandRegio to Zensus grid
zensus_population ()
    Bring population prognosis from DemandRegio to Zensus grid

```

10.5.25 substation_voronoi

The central module containing code to create substation voronois

```

class EgonEhvSubstationVoronoi (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus_id
    geom
    id
class EgonHvmvSubstationVoronoi (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus_id
    geom
    id
class SubstationVoronoi (dependencies)
    Bases: egon.data.datasets.Dataset
create_tables ()
    Create tables for voronoi polygons :returns: None.
substation_voronoi ()
    Creates voronoi polygons for hvmv and ehv substations
    Returns None.

```

10.5.26 tyndp

The central module containing all code dealing with downloading tyndp data

```

class Tyndp (dependencies)
    Bases: egon.data.datasets.Dataset
    Downloads data for foreign countries from Ten-Year-Network-Development Plan
    This dataset downloads installed generation capacities and load time series for foreign countries from the website
    of the Ten-Year-Network-Development Plan 2020 from ENTSO-E. That data is stored into files and later on
    written into the database (see ElectricalNeighbours).

```

Dependencies

- Setup

Resulting tables**name** = 'Tyndp'**version** = '0.0.1'**download()**Download input data from TYNDP 2020 :returns: *None*.

10.5.27 vg250_mv_grid_districts

The module containing all code dealing with pv rooftop distribution.

class MapMvgriddistrictsVg250 (**kwargs)

Bases: sqlalchemy.ext.declarative.api.Base

bus_id**vg250_lan****class Vg250MvGridDistricts** (dependencies)Bases: *egon.data.datasets.Dataset***create_tables()**Create tables for mapping grid districts to federal state :returns: *None*.**mapping()**

Map mv grid districts to federal states

Returns *None*.

10.5.28 zensus_mv_grid_districts

Implements mapping between mv grid districts and zensus cells

class MapZensusGridDistricts (**kwargs)

Bases: sqlalchemy.ext.declarative.api.Base

bus_id**zensus_population_id****class ZensusMvGridDistricts** (dependencies)Bases: *egon.data.datasets.Dataset***mapping()**

Perform mapping between mv grid districts and zensus grid

10.5.29 zensus_vg250

class DestatisZensusPopulationPerHa (**kwargs)

Bases: sqlalchemy.ext.declarative.api.Base

geom**geom_point**

```
    grid_id
    id
    population
    x_mp
    y_mp
class DestatisZensusPopulationPerHaInsideGermany (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    geom
    geom_point
    grid_id
    id
    population
class MapZensusVg250 (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    vg250_municipality_id
    vg250_nuts3
    zensus_geom
    zensus_population_id
class Vg250Gem (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    ade
    ags
    ags_0
    ars
    ars_0
    bem
    bez
    bsg
    debkg_id
    fk_s3
    gen
    geometry
    gf
    ibz
    id
    nbd
    nuts
```

```
rs
rs_0
sdv_ars
sdv_rs
sn_g
sn_k
sn_l
sn_r
sn_v1
sn_v2
wsk

class Vg250GemPopulation(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    ags_0
    area_ha
    area_km2
    bem
    bez
    cell_count
    gen
    geom
    id
    nuts
    population_density
    population_total
    rs_0

class Vg250Sta(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    ade
    ags
    ags_0
    ars
    ars_0
    bem
    bez
    bsg
    debkg_id
```



```

    fk_s3
    gen
    geometry
    gf
    ibz
    id
    nbd
    nuts
    rs
    rs_0
    sdv_ars
    sdv_rs
    sn_g
    sn_k
    sn_l
    sn_r
    sn_v1
    sn_v2
    wsk

class ZensusVg250 (dependencies)
    Bases: egon.data.datasets.Dataset

add_metadata_vg250_gem_pop ()
    Create metadata JSON for Vg250GemPopulation
    Creates a metdadadata JSON string and writes it to the database table comment

add_metadata_zensus_inside_ger ()
    Create metadata JSON for DestatisZensusPopulationPerHaInsideGermany
    Creates a metdadadata JSON string and writes it to the database table comment

inside_germany ()
    Filter zensus data by data inside Germany and population > 0

map_zensus_vg250 ()
    Perform mapping between municipalities and zensus grid

population_in_municipalities ()
    Create table of municipalities with information about population

```

10.5.30 chp

match_nep

The module containing all code dealing with large chp from NEP list.

insert_large_chp (*sources, target, EgonChp*)

match_nep_chp (*chp_NEP, MaStR_konv, chp_NEP_matched, buffer_capacity=0.1, consider_location='plz', consider_carrier=True, consider_capacity=True*)
Match CHP plants from MaStR to list of power plants from NEP

Parameters

- **chp_NEP** (*pandas.DataFrame*) – CHP plants from NEP which are not matched to MaStR
- **MaStR_konv** (*pandas.DataFrame*) – CHP plants from MaStR which are not matched to NEP
- **chp_NEP_matched** (*pandas.DataFrame*) – Already matched CHP
- **buffer_capacity** (*float, optional*) – Maximum difference in capacity in p.u. The default is 0.1.

Returns

- **chp_NEP_matched** (*pandas.DataFrame*) – Matched CHP
- **MaStR_konv** (*pandas.DataFrame*) – CHP plants from MaStR which are not matched to NEP
- **chp_NEP** (*pandas.DataFrame*) – CHP plants from NEP which are not matched to MaStR

select_chp_from_mastr (*sources*)

Select combustion CHP plants from MaStR

Returns **MaStR_konv** (*pd.DataFrame*) – CHP plants from MaStR

select_chp_from_nep (*sources*)

Select CHP plants with location from NEP's list of power plants

Returns *pandas.DataFrame* – CHP plants from NEP list

small_chp

The module containing all code dealing with *chp* < 10MW.

assign_use_case (*chp, sources*)

Identifies CHPs used in district heating areas.

A CHP plant is assigned to a district heating area if - it is closer than 1km to the borders of the district heating area - the name of the osm landuse area where the CHP is located indicates that it feeds in to a district heating area (e.g. 'Stadtwerke') - it is not closer than 100m to an industrial area

Parameters **chp** (*pandas.DataFrame*) – CHPs without *district_heating* flag

Returns **chp** (*pandas.DataFrame*) – CHPs with identification of *district_heating* CHPs

existing_chp_smaller_10mw (*sources, MaStR_konv, EgonChp*)

Insert existing small CHPs based on MaStR and target values

Parameters

- **MaStR_konv** (*pandas.DataFrame*) – List of conventional CHPs in MaStR whose location is not used
- **EgonChp** (*class*) – Class definition of database table for CHPs

Returns **additional_capacity** (*pandas.Series*) – Capacity of new locations for small *chp* per federal state

extension_district_heating (*federal_state, additional_capacity, flh_chp, EgonChp, areas_without_chp_only=False*)

Build new CHP < 10 MW for district areas considering existing CHP and the heat demand.

For more details on the placement algorithm have a look at the description of `extension_to_areas()`.

Parameters

- **federal_state** (*str*) – Name of the federal state.
- **additional_capacity** (*float*) – Additional electrical capacity of new CHP plants in district heating
- **flh_chp** (*int*) – Assumed number of full load hours of heat output.
- **EgonChp** (*class*) – ORM-class definition of CHP database-table.
- **areas_without_chp_only** (*boolean, optional*) – Set if CHPs are only assigned to district heating areas which don't have an existing CHP. The default is True.

Returns *None*.

extension_industrial (*federal_state, additional_capacity, flh_chp, EgonChp*)

Build new CHP < 10 MW for industry considering existing CHP, osm landuse areas and electricity demands.

For more details on the placement algorithm have a look at the description of `extension_to_areas()`.

Parameters

- **federal_state** (*str*) – Name of the federal state.
- **additional_capacity** (*float*) – Additional electrical capacity of new CHP plants in industry.
- **flh_chp** (*int*) – Assumed number of full load hours of electricity output.
- **EgonChp** (*class*) – ORM-class definition of CHP database-table.

Returns *None*.

extension_per_federal_state (*federal_state, EgonChp*)

Adds new CHP plants to meet target value per federal state.

The additional capacity for CHPs < 10 MW is distributed discretely. Therefore, existing CHPs and their parameters from Marktstammdatenregister are randomly selected and allocated in a district heating grid. In order to generate a reasonable distribution, new CHPs can only be assigned to a district heating grid which needs additional supply technologies. This is estimated by the subtraction of demand, and the assumed dispatch of a CHP considering the capacity and full load hours of each CHPs.

Parameters

- **additional_capacity** (*float*) – Capacity to distribute.
- **federal_state** (*str*) – Name of the federal state
- **EgonChp** (*class*) – ORM-class definition of CHP table

Returns *None*.

extension_to_areas (*areas, additional_capacity, existing_chp, flh, EgonChp, district_heating=True, scenario='eGon2035'*)

Builds new CHPs on potential industry or district heating areas.

This method can be used to discretely extend and spatially allocate CHP for industry or district heating areas. The following steps are running in a loop until the additional capacity is reached:

1. Randomly select an existing CHP < 10MW and its parameters.
2. Select possible areas where the CHP can be located. It is assumed that CHPs are only build if the demand of the industry or district heating grid exceeds the annual energy output of the CHP. The energy output is calculated using the installed capacity and estimated full load hours. The thermal output is used for district heating areas. Since there are no explicit heat demands for industry, the electricity output and demands are used.
3. Randomly select one of the possible areas. The areas are weighted by the annal demand, assuming that the possibility of building a CHP plant is higher when for large consumers.
4. Insert allocated CHP plant into the database
5. Substract capacity of new build CHP from the additional capacity. The energy demands of the areas are reduced by the estimated energy output of the CHP plant.

Parameters

- **areas** (*geopandas.GeoDataFrame*) – Possible areas for a new CHP plant, including their energy demand
- **additional_capacity** (*float*) – Overall eletrical capacity of CHPs that should be build in MW.
- **existing_chp** (*pandas.DataFrame*) – List of existing CHP plants including electrical and thermal capacity
- **flh** (*int*) – Assumed electrical or thermal full load hours.
- **EgonChp** (*class*) – ORM-class definition of CHP database-table.
- **district_heating** (*boolean, optional*) – State if the areas are district heating areas. The default is True.

Returns *None*.

insert_mastr_chp (*mastr_chp, EgonChp*)

Insert MaStR data from existing CHPs into database table

Parameters

- **mastr_chp** (*pandas.DataFrame*) – List of existing CHPs in MaStR.
- **EgonChp** (*class*) – Class definition of daabase table for CHPs

Returns *None*.

The central module containing all code dealing with combined heat and power (CHP) plants.

class Chp (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Extract combined heat and power plants for each scenario

This dataset creates combined heat and power (CHP) plants for each scenario and defines their use case. The method bases on existing CHP plants from Marktstammdatenregister. For the eGon2035 scenario, a list of CHP plans from the grid operator is used for new largescale CHP plants. CHP < 10MW are randomly distributed. Depending on the distance to a district heating grid, it is decided if the CHP is used to supply a district heating grid or used by an industrial site.

Dependencies

- *GasAreaseGon100RE*
- *GasAreaseGon2035*

- *DistrictHeatingAreas*
- *IndustrialDemandCurves*
- *OsmLanduse*
- *download_mastr_data*
- *define_mv_grid_districts*
- *ScenarioCapacities*

Resulting tables

- *supply.egon_chp_plants* is created and filled
- *supply.egon_mastr_conventional_without_chp* is created and filled

```

name = 'Chp'
version = '0.0.6'

class EgonChp(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    carrier
    ch4_bus_id
    district_heating
    district_heating_area_id
    el_capacity
    electrical_bus_id
    geom
    id
    scenario
    source_id
    sources
    th_capacity
    voltage_level

class EgonMaStRConventinalWithoutChp(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    EinheitMastrNummer
    carrier
    city
    el_capacity
    federal_state
    geometry
    id
    plz

```

assign_heat_bus (*scenario='eGon2035'*)

Selects heat_bus for chps used in district heating.

Parameters *scenario* (*str; optional*) – Name of the corresponding scenario. The default is 'eGon2035'.

Returns *None*.

create_tables ()

Create tables for chp data :returns: *None*.

extension_BB ()

extension_BE ()

extension_BW ()

extension_BY ()

extension_HB ()

extension_HE ()

extension_HH ()

extension_MV ()

extension_NS ()

extension_NW ()

extension_RP ()

extension_SH ()

extension_SL ()

extension_SN ()

extension_ST ()

extension_TH ()

insert_biomass_chp (*scenario*)

Insert biomass chp plants of future scenario

Parameters *scenario* (*str*) – Name of scenario.

Returns *None*.

insert_chp_egon100re ()

Insert CHP plants for eGon100RE considering results from pypsa-eur-sec

Returns *None*.

insert_chp_egon2035 ()

Insert CHP plants for eGon2035 considering NEP and MaStR data

Returns *None*.

nearest (*row*, *df*, *centroid=False*, *row_geom_col='geometry'*, *df_geom_col='geometry'*,
src_column=None)

Finds the nearest point and returns the specified column values

Parameters

- **row** (*pandas.Series*) – Data to which the nearest data of df is assigned.

- **df** (*pandas.DataFrame*) – Data which includes all options for the nearest neighbor algorithm.
- **centroid** (*boolean*) – Use centroid geometry. The default is False.
- **row_geom_col** (*str, optional*) – Name of row's geometry column. The default is 'geometry'.
- **df_geom_col** (*str, optional*) – Name of df's geometry column. The default is 'geometry'.
- **src_column** (*str, optional*) – Name of returned df column. The default is None.

Returns *value* (*pandas.Series*) – Values of specified column of df

10.5.31 data_bundle

The central module containing all code dealing with small scale input data

```
class DataBundle (dependencies)
    Bases: egon.data.datasets.Dataset

download ()
    Download small scale input data from Zenodo
```

10.5.32 demandregio

install_disaggregator

This module downloads and installs demandregio's disaggregator from GitHub

```
clone_and_install ()
    Clone and install repository of demandregio's disaggregator

Returns None.
```

The central module containing all code dealing with importing and adjusting data from demandRegio

```
class DemandRegio (dependencies)
    Bases: egon.data.datasets.Dataset

class EgonDemandRegioCtsInd (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    demand
    nuts3
    scenario
    wz
    year

class EgonDemandRegioHH (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    demand
    hh_size
    nuts3
    scenario
```

```
year
```

```
class EgonDemandRegioHouseholds (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    hh_size

    households

    nuts3

    year
```

```
class EgonDemandRegioPopulation (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    nuts3

    population

    year
```

```
class EgonDemandRegioTimeseriesCtsInd (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    load_curve

    slp

    wz

    year
```

```
class EgonDemandRegioWz (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    definition

    sector

    wz
```

adjust_cts_ind_nep (*ec_cts_ind*, *sector*)
Add electrical demand of new largescale CTS und industrial consumers according to NEP 2021, scenario C 2035. Values per federal state are linear distributed over all CTS branches and nuts3 regions.

Parameters *ec_cts_ind* (*pandas.DataFrame*) – CTS or industry demand without new largescale consumers.

Returns *ec_cts_ind* (*pandas.DataFrame*) – CTS or industry demand including new largescale consumers.

adjust_ind_pes (*ec_cts_ind*)
Adjust electricity demand of industrial consumers due to electrification of process heat based on assumptions of pypsa-eur-sec.

Parameters *ec_cts_ind* (*pandas.DataFrame*) – Industrial demand without additional electrification

Returns *ec_cts_ind* (*pandas.DataFrame*) – Industrial demand with additional electrification

create_tables ()
Create tables for demandregio data :returns: *None*.

data_in_boundaries (*df*)
Select rows with nuts3 code within boundaries, used for testmode

Parameters *df* (*pandas.DataFrame*) – Data for all nuts3 regions

Returns *pandas.DataFrame* – Data for nuts3 regions within boundaries

disagg_households_power (*scenario, year, weight_by_income=False, original=False, **kwargs*)

Perform spatial disaggregation of electric power in [GWh/a] by key and possibly weight by income. Similar to `disagggregator.spatial.disagg_households_power`

Parameters

- **by** (*str*) – must be one of ['households', 'population']
- **weight_by_income** (*bool, optional*) – Flag if to weight the results by the regional income (default False)
- **original** (*bool, optional*) – Throughput to function `households_per_size`, A flag if the results should be left untouched and returned in original form for the year 2011 (True) or if they should be scaled to the given *year* by the population in that year (False).

Returns *pd.DataFrame* or *pd.Series*

insert_cts_ind (*scenario, year, engine, target_values*)

Calculates electrical demands of CTS and industry using `demandregio`'s disaggregator, adjusts them according to resulting values of NEP 2021 or JRC IDEES and insert results into the database.

Parameters

- **scenario** (*str*) – Name of the corresponding scenario.
- **year** (*int*) – The number of households per region is taken from this year.
- **target_values** (*dict*) – List of target values for each scenario and sector.

Returns *None*.

insert_cts_ind_demands ()

Insert electricity demands per nuts3-region in Germany according to `demandregio` using its disaggregator-tool in MWh

Returns *None*.

insert_cts_ind_wz_definitions ()

Insert `demandregio`'s definitions of CTS and industrial branches

Returns *None*.

insert_hh_demand (*scenario, year, engine*)

Calculates electrical demands of private households using `demandregio`'s disaggregator and insert results into the database.

Parameters

- **scenario** (*str*) – Name of the corresponding scenario.
- **year** (*int*) – The number of households per region is taken from this year.

Returns *None*.

insert_household_demand ()

Insert electrical demands for households according to `demandregio` using its disaggregator-tool in MWh

Returns *None*.

insert_society_data ()

Insert population and number of households per nuts3-region in Germany according to `demandregio` using its disaggregator-tool

Returns *None*.

insert_timeseries_per_wz (*sector*, *year*)

Insert normalized electrical load time series for the selected sector

Parameters

- **sector** (*str*) – Name of the sector. ['CTS', 'industry']
- **year** (*int*) – Selected weather year

Returns *None*.

match_nuts3_b1 ()

Function that maps the federal state to each nuts3 region

Returns **df** (*pandas.DataFrame*) – List of nuts3 regions and the federal state of Germany.

timeseries_per_wz ()

Calculate and insert normalized timeseries per wz for cts and industry

Returns *None*.

10.5.33 district_heating_areas

plot

Module containing all code creating with plots of district heating areas

plot_heat_density_sorted (*heat_density_per_scenario*, *scenario_name=None*)

Create diagrams for visualisation, sorted by HDD sorted census dh first, sorted new areas, left overs, DH share
create one dataframe with all data: first the cells with existing, then the cells with new district heating systems
and in the end the ones without

Parameters

- **scenario_name** (*TYPE*) – DESCRIPTION.
- **collection** (*TYPE*) – DESCRIPTION.

Returns *None*.

Central module containing all code creating with district heating areas.

This module obtains the information from the census tables and the heat demand densities, demarcates so the current and future district heating areas. In the end it saves them in the database.

class DistrictHeatingAreas (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Create district heating grids for all scenarios

This dataset creates district heating grids for each scenario based on a defined district heating share, annual heat demands calculated within *HeatDemandImport* and information on existing heating grids from census *ZensusMiscellaneous*

First the tables are created using *create_tables()*. Afterwards, the district heating grids for each scenario are created and inserted into the database by applying the function *district_heating_areas()*

Dependencies

- *HeatDemandImport*
- *ZensusMiscellaneous*
- *ScenarioParameters*

Resulting tables

- `demand.egon_map_zensus_district_heating_areas` is created and filled
- `demand.egon_district_heating_areas` is created and filled

```
name = 'district-heating-areas'
```

```
version = '0.0.1'
```

```
class EgonDistrictHeatingAreas (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
```

```
    area_id
```

```
    geom_polygon
```

```
    id
```

```
    residential_and_service_demand
```

```
    scenario
```

```
class MapZensusDistrictHeatingAreas (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
```

```
    area_id
```

```
    id
```

```
    scenario
```

```
    zensus_population_id
```

```
add_metadata ()
```

Writes metadata JSON string into table comment.

```
area_grouping (raw_polygons, distance=200, minimum_total_demand=None, maxi-
                mum_total_demand=None)
```

Group polygons which are close to each other.

This function creates buffers around the given cell polygons (called “raw_polygons”) and unions the intersecting buffer polygons. Afterwards, it unions the cell polygons which are within one unified buffer polygon. If requested, the cells being in areas fulfilling the minimum heat demand criterium are selected.

Parameters

- **raw_polygons** (*geopandas.geodataframe.GeoDataFrame*) – polygons to be grouped.
- **distance** (*integer*) – distance for buffering
- **minimum_total_demand** (*integer*) – optional minimum total heat demand to achieve a minimum size of areas
- **maximal_total_demand** (*integer*) – optional maximal total heat demand per area, if demand is higher the area is cut at nuts3 borders

Returns `join (geopandas.geodataframe.GeoDataFrame)` – cell polygons with area id

Notes

None

```
create_tables ()
```

Create tables for district heating areas

Returns *None*

demarcation (*plotting=True*)

Load scenario specific district heating areas with metadata into database.

This function executes the functions that identifies the areas which will be supplied with district heat in the two eGoⁿ scenarios. The creation of heat demand density curve figures is optional. So is also the export of scenario specific Prospective Supply Districts for district heating (PSDs) as shapefiles including the creation of a figure showing the comparison of sorted heat demand densities.

The method was executed for 2015, 2035 and 2050 to find out which scenario year defines the PSDs. The year 2035 was selected and the function was adjusted accordingly. If you need the 2015 scenario heat demand data, please have a look at the heat demand script commit 270bea50332016447e869f69d51e96113073b8a0, where the 2015 scenario was deactivated. You can study the 2015 PSDs in the `study_prospective_district_heating_areas` function after un-commenting some lines.

Parameters **plotting** (*boolean*) – if True, figure showing the heat demand density curve will be created

Returns *None*

Notes

None

district_heating_areas (*scenario_name, plotting=False*)

Create scenario specific district heating areas considering on census data.

This function loads the district heating share from the scenario table and demarcate the scenario specific district heating areas. To do so it uses the census data on flats currently supplied with district heat, which are supplied selected first, if the estimated connection rate $\geq 30\%$.

All scenarios use the Prospective Supply Districts (PSDs) made for the eGon2035 scenario to identify the areas where additional district heating supply is feasible. One PSD dataset is to defined which is constant over the years to allow comparisons. Moreover, it is assumed that the eGon2035 PSD dataset is suitable, even though the heat demands will continue to decrease from 2035 to 2050, because district heating systems will be to planned and built before 2050, to exist in 2050.

It is assumed that the connection rate in cells with district heating will be a 100%. That is because later in project the number of buildings per cell will be used and connection rates not being 0 or 100% will create buildings which are not fully supplied by one technology.

The cell polygons which carry information (like heat demand etc.) are grouped into areas which are close to each other. Only cells with a minimum heat demand density (e.g. $>100 \text{ GJ}/(\text{ha a})$) are considered when creating PSDs. Therefore, the `select_high_heat_demands()` function is used. There is minimum heat demand per PSDs to achieve a certain size. While the grouping buffer for the creation of Prospective Supply Districts (PSDs) is 200m as in the sEEnergies project, the buffer for grouping census data cell with an estimated connection rate $\geq 30\%$ is 500m. The 500m buffer is also used when the resulting district heating areas are grouped, because they are built upon the existing district heating systems.

To reduce the final number of district heating areas having the size of only one hectare, the minimum heat demand critrium is also applied when grouping the cells with census data on district heat.

To avoid huge district heating areas, as they appear in the Ruhr area, district heating areas with an annual demand $> 4,000,000 \text{ MWh}$ are split by nuts3 boundaries. This as set as `maximum_total_demand` of the `area_grouping` function.

Parameters

- **scenario_name** (*str*) – name of scenario to be studies

- **plotting** (*boolean*) – if True, figure showing the heat demand density curve will be created

Returns *None*

Notes

None

load_census_data ()

Load the heating type information from the census database table.

The census apartment and the census building table contains information about the heating type. The information are loaded from the apartment table, because they might be more useful when it comes to the estimation of the connection rates. Only cells with a connection rate equal to or larger than 30% (based on the census apartment data) are included in the returned `district_heat` GeoDataFrame.

Parameters *None*

Returns

- **district_heat** (*geopandas.geodataframe.GeoDataFrame*) – polygons (hectare cells) with district heat information
- **heating_type** (*geopandas.geodataframe.GeoDataFrame*) – polygons (hectare cells) with the number of flats having heating type information

Notes

The census contains only information on residential buildings. Therefore, also connection rate of the residential buildings can be estimated.

load_heat_demands (*scenario_name*)

Load scenario specific heat demand data from the local database.

Parameters **scenario_name** (*str*) – name of the scenario studied

Returns **heat_demand** (*geopandas.geodataframe.GeoDataFrame*) – polygons (hectare cells) with heat demand data

select_high_heat_demands (*heat_demand*)

Take heat demand cells and select cells with higher heat demand.

Those can be used to identify prospective district heating supply areas.

Parameters **heat_demand** (*geopandas.geodataframe.GeoDataFrame*) – dataset of heat demand cells.

Returns **high_heat_demand** (*geopandas.geodataframe.GeoDataFrame*) – polygons (hectare cells) with heat demands high enough to be potentially high enough to be in a district heating area

study_prospective_district_heating_areas ()

Get information about Prospective Supply Districts for district heating.

This optional function executes the functions so that you can study the heat demand density data of different scenarios and compare them and the resulting Prospective Supply Districts (PSDs) for district heating. This functions saves local shapefiles, because these data are not written into database. Moreover, heat density curves are drawn. This function is tailor-made and includes the scenarios eGon2035 and eGon100RE.

Parameters *None*

Returns *None*

Notes

None

10.5.34 electricity_demand

temporal

The central module containing all code dealing with processing timeseries data using demandregio

```
class EgonEtragoElectricityCts (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    bus_id
    p_set
    q_set
    scn_name
```

```
calc_load_curve (share_wz, annual_demand=1)
    Create aggregated demand curve for service sector
```

Parameters

- **share_wz** (*pandas.Series or pandas.DataFrame*) – Share of annual demand per cts branch
- **annual_demand** (*float or pandas.Series, optional*) – Annual demand in MWh. The default is 1.

Returns *pandas.Series or pandas.DataFrame* – Annual load curve of combines cts branches

```
calc_load_curves_cts (scenario)
    Temporal disaggregate electrical cts demand per substation.
```

Parameters **scenario** (*str*) – Scenario name.

Returns *pandas.DataFrame* – Demand timeseries of cts per bus id

```
create_table ()
    Create tables for demandregio data :returns: None.
```

```
insert_cts_load ()
    Inserts electrical cts loads to etrago-tables in the database
```

Returns *None*.

The central module containing all code dealing with processing data from demandRegio

```
class CtsElectricityDemand (dependencies)
    Bases: egon.data.datasets.Dataset

class EgonDemandRegioZensusElectricity (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    demand
    scenario
    sector
    zensus_population_id
```

class HouseholdElectricityDemand (*dependencies*)

Bases: *egon.data.datasets.Dataset*

create_tables ()

Create tables for demandregio data :returns: *None*.

distribute_cts_demands ()

Distribute electrical demands for cts to zensus cells.

The demands on nuts3-level from demandregio are linear distributed to the heat demand of cts in each zensus cell.

Returns *None*.

get_annual_household_el_demand_cells ()

Annual electricity demand per cell is determined

Timeseries for every cell are accumulated, the maximum value determined and with the respective nuts3 factor scaled for 2035 and 2050 scenario.

Note: In test-mode 'SH' the iteration takes place by 'cell_id' to avoid intensive RAM usage. For whole Germany 'nuts3' are taken and RAM > 32GB is necessary.

10.5.35 electricity_demand_timeseries

cts_buildings

CTS electricity and heat demand time series for scenarios in 2035 and 2050 assigned to OSM-buildings.

Disaggregation of cts heat & electricity demand time series from MV Substation to census cells via annual demand and then to OSM buildings via amenity tags or randomly if no sufficient OSM-data is available in the respective census cell. If no OSM-buildings or synthetic residential buildings are available new synthetic 5x5m buildings are generated.

The resulting data is stored in separate tables

- **openstreetmap.osm_buildings_synthetic:** Lists generated synthetic building with id, zensus_population_id and building type. This table is already created within `hh_buildings.map_houseprofiles_to_buildings()`
- **openstreetmap.egon_cts_buildings:** Table of all selected cts buildings with id, census cell id, geometry and amenity count in building. This table is created within `cts_buildings()`
- **demand.egon_cts_electricity_demand_building_share:** Table including the mv substation electricity profile share of all selected cts buildings for scenario eGon2035 and eGon100RE. This table is created within `cts_electricity()`
- **demand.egon_cts_heat_demand_building_share:** Table including the mv substation heat profile share of all selected cts buildings for scenario eGon2035 and eGon100RE. This table is created within `cts_heat()`
- **demand.egon_building_electricity_peak_loads:** Mapping of electricity demand time series and buildings including cell_id, building area and peak load. This table is already created within `hh_buildings.get_building_peak_loads()`
- **boundaries.egon_map_zensus_mvgd_buildings:** A final mapping table including all buildings used for residential and cts, heat and electricity timeseries. Including census cells, mvgd bus_id, building type (osm or synthetic)

The following datasets from the database are mainly used for creation:

- ***openstreetmap.osm_buildings_filtered***: Table of OSM-buildings filtered by tags to selecting residential and cts buildings only.
- ***openstreetmap.osm_amenities_shops_filtered***: Table of OSM-amenities filtered by tags to select cts only.
- ***openstreetmap.osm_amenities_not_in_buildings_filtered***: Table of amenities which do not intersect with any building from *openstreetmap.osm_buildings_filtered*
- ***openstreetmap.osm_buildings_synthetic***: Table of synthetic residential buildings
- ***boundaries.egon_map_zensus_buildings_filtered_all***: Mapping table of census cells and buildings filtered even if population in census cell = 0.
- ***demand.egon_demandregio_zensus_electricity***: Table of annual electricity load demand for residential and cts at census cell level. Residential load demand is derived from aggregated residential building profiles. DemandRegio CTS load demand at NUTS3 is distributed to census cells linearly to heat demand from *peta5*.
- ***demand.egon_peta_heat***: Table of annual heat load demand for residential and cts at census cell level from *peta5*.
- ***demand.egon_etrage_electricity_cts***: Scaled cts electricity time series for every MV substation. Derived from DemandRegio SLP for selected economic sectors at nuts3. Scaled with annual demand from *demand.egon_demandregio_zensus_electricity*
- ***demand.egon_etrage_heat_cts***: Scaled cts heat time series for every MV substation. Derived from DemandRegio SLP Gas for selected economic sectors at nuts3. Scaled with annual demand from *demand.egon_peta_heat*.

What is the goal?

To disaggregate cts heat and electricity time series from MV substation level to geo-referenced buildings, the annual demand from DemandRegio and Peta5 is used to identify census cells with load demand. We use Openstreetmap data and filter tags to identify buildings and count the amenities within. The number of amenities and the annual demand serve to assign a demand share of the MV substation profile to the building.

What is the challenge?

The OSM, DemandRegio and Peta5 dataset differ from each other. The OSM dataset is a community based dataset which is extended throughout and does not claim to be complete. Therefore, not all census cells which have a demand assigned by DemandRegio or Peta5 methodology also have buildings with respective tags or sometimes even any building at all. Furthermore, the substation load areas are determined dynamically in a previous dataset. Merging these datasets different scopes (census cell shapes, building shapes) and their inconsistencies need to be addressed. For example: not yet tagged buildings or amenities in OSM, or building shapes exceeding census cells.

How are these datasets combined?

The methodology for heat and electricity is the same and only differs in the annual demand and MV/HV Substation profile. In a previous dataset (*openstreetmap*), we filter all OSM buildings and amenities for tags, we relate to the cts sector. Amenities are mapped to intersecting buildings and then intersected with the annual demand which exists at census cell level. We obtain census cells with demand and amenities and without amenities. If there is no data on amenities, *n* synthetic ones are assigned to existing buildings. We use the median value of amenities/census cell for *n* and all filtered buildings + synthetic residential buildings. If no building data is available a synthetic buildings is randomly generated. This also happens for amenities which couldn't be assigned to any osm building. All census cells with an annual demand are covered this way, and we obtain four different categories of buildings with amenities:

- Buildings with amenities
- Synthetic buildings with amenities
- Buildings with synthetic amenities
- Synthetics buildings with synthetic amenities

The amenities are summed per census cell (of amenity) and building to derive the building amenity share per census cell. Multiplied with the annual demand, we receive the profile demand share for each cell. Some buildings exceed the census cell shape and have amenities in different cells although mapped to one building only. To have unique buildings the demand share is summed once more per building id. This factor can now be used to obtain the profile for each building.

A schematic flow chart exist in the correspondent issue #671: <https://github.com/openego/eGon-data/issues/671#issuecomment-1260740258>

What are central assumptions during the data processing?

- We assume OSM data to be the most reliable and complete open source dataset.
- We assume building and amenity tags to be truthful and accurate.
- Mapping census to OSM data is not trivial. Discrepancies are substituted.
- Missing OSM buildings are generated for each amenity.
- Missing amenities are generated by median value of amenities/census cell.

Drawbacks and limitations of the data

- Shape of profiles for each building is similar within a MVGD and only scaled

with a different factor. * MVGDs are generated dynamically. In case of buildings with amenities exceeding MVGD borders, amenities which are assigned to a different MVGD than the assigned building centroid, the amenities are dropped for sake of simplicity. One building should not have a connection to two MVGDs. * The completeness of the OSM data depends on community contribution and is crucial to the quality of our results. * Randomly selected buildings and generated amenities may inadequately reflect reality, but are chosen for sake of simplicity as a measure to fill data gaps. * Since this dataset is a cascade after generation of synthetic residential buildings also check drawbacks and limitations in `hh_buildings.py`. * Synthetic buildings may be placed within osm buildings which exceed multiple census cells. This is currently accepted but may be solved in #953 * Scattered high peak loads occur and might lead to single MV grid connections in `ding0`. In some cases this might not be viable. Postprocessing is needed and may be solved in #954.

Example Query

Notes

This module docstring is rather a dataset documentation. Once, a decision is made in ... the content of this module docstring needs to be moved to docs attribute of the respective dataset class.

```
class BuildingHeatPeakLoads (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    building_id
    peak_load_in_w
    scenario
    sector

class CtsBuildings (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    geom_building
    id
    n_amenities_inside
```

```
serial
source
zensus_population_id
```

```
class CtsDemandBuildings (dependencies)
    Bases: egon.data.datasets.Dataset
```

```
class EgonCtsElectricityDemandBuildingShare (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    building_id
    bus_id
    profile_share
    scenario
```

```
class EgonCtsHeatDemandBuildingShare (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    building_id
    bus_id
    profile_share
    scenario
```

amenities_without_buildings ()
Amenities which have no buildings assigned and are in a cell with cts demand are determined.

Returns *pd.DataFrame* – Table of amenities without buildings

assign_voltage_level_to_buildings ()
Add voltage level to all buildings by summed peak demand.

All entries with same building id get the voltage level corresponding to their summed residential and cts peak demand.

buildings_with_amenities ()
Amenities which are assigned to buildings are determined and grouped per building and zensus cell. Buildings covering multiple cells therefore exists multiple times but in different zensus cells. This is necessary to cover as many cells with a cts demand as possible. If buildings exist in multiple mvgs (bus_id) , only the amenities within the same as the building centroid are kept. If as a result, a census cell is uncovered by any buildings, a synthetic amenity is placed. The buildings are aggregated afterwards during the calculation of the profile_share.

Returns

- **df_buildings_with_amenities** (*gpd.GeoDataFrame*) – Contains all buildings with amenities per zensus cell.
- **df_lost_cells** (*gpd.GeoDataFrame*) – Contains synthetic amenities in lost cells. Might be empty

buildings_without_amenities ()
Buildings (filtered and synthetic) in cells with cts demand but no amenities are determined.

Returns **df_buildings_without_amenities** (*gpd.GeoDataFrame*) – Table of buildings without amenities in zensus cells with cts demand.

```
calc_building_demand_profile_share (df_cts_buildings,          scenario='eGon2035',          sec-
                                     tor='electricity')
```

Share of cts electricity demand profile per bus for every selected building is calculated. Building-amenity share

is multiplied with census cell share to get the substation bus profile share for each building. The share is grouped and aggregated per building as some buildings exceed the shape of census cells and have amenities assigned from multiple cells. Building therefore get the amenity share of all census cells.

Parameters

- **df_cts_buildings** (*gpd.GeoDataFrame*) – Table of all buildings with cts demand assigned
- **scenario** (*str*) – Scenario for which the share is calculated.
- **sector** (*str*) – Sector for which the share is calculated.

Returns **df_building_share** (*pd.DataFrame*) – Table of bus profile share per building

calc_census_cell_share (*scenario, sector*)

The profile share for each census cell is calculated by it's share of annual demand per substation bus. The annual demand per cell is defined by DemandRegio/Peta5. The share is for both scenarios identical as the annual demand is linearly scaled.

Parameters

- **scenario** (*str*) – Scenario for which the share is calculated: “eGon2035” or “eGon100RE”
- **sector** (*str*) – Scenario for which the share is calculated: “electricity” or “heat”

Returns **df_census_share** (*pd.DataFrame*)

calc_cts_building_profiles (*bus_ids, scenario, sector*)

Calculate the cts demand profile for each building. The profile is calculated by the demand share of the building per substation bus.

Parameters

- **bus_ids** (*list of int*) – Ids of the substation for which selected building profiles are calculated.
- **scenario** (*str*) – Scenario for which the share is calculated: “eGon2035” or “eGon100RE”
- **sector** (*str*) – Sector for which the share is calculated: “electricity” or “heat”

Returns **df_building_profiles** (*pd.DataFrame*) – Table of demand profile per building. Column names are building IDs and index is hour of the year as int (0-8759).

cells_with_cts_demand_only (*df_buildings_without_amenities*)

Cells with cts demand but no amenities or buildings are determined.

Returns **df_cells_only_cts_demand** (*gpd.GeoDataFrame*) – Table of cells with cts demand but no amenities or buildings

create_synthetic_buildings (*df, points=None, crs='EPSG:3035'*)

Synthetic buildings are generated around points.

Parameters

- **df** (*pd.DataFrame*) – Table of census cells
- **points** (*gpd.GeoSeries or str*) – List of points to place buildings around or column name of df
- **crs** (*str*) – CRS of result table

Returns **df** (*gpd.GeoDataFrame*) – Synthetic buildings

cts_buildings ()

Assigns CTS demand to buildings and calculates the respective demand profiles. The demand profile per substation are disaggregated per annual demand share of each census cell and by the number of amenities per building within the cell. If no building data is available, synthetic buildings are generated around the amenities. If no

amenities but cts demand is available, buildings are randomly selected. If no building nor amenity is available, random synthetic buildings are generated. The demand share is stored in the database.

Cells with CTS demand, amenities and buildings do not change within the scenarios, only the demand itself. Therefore scenario eGon2035 can be used universally to determine the cts buildings but not for the demand share.

cts_electricity()

Calculate cts electricity demand share of hvmv substation profile for buildings.

cts_heat()

Calculate cts electricity demand share of hvmv substation profile for buildings.

delete_synthetic_cts_buildings()

All synthetic cts buildings are deleted from the DB. This is necessary if the task is run multiple times as the existing synthetic buildings influence the results.

get_cts_electricity_peak_load()

Get electricity peak load of all CTS buildings for both scenarios and store in DB.

get_cts_heat_peak_load()

Get heat peak load of all CTS buildings for both scenarios and store in DB.

get_peta_demand(mvgd, scenario)

Retrieve annual peta heat demand for CTS for either eGon2035 or eGon100RE scenario.

Parameters

- **mvgd** (*int*) – ID of substation for which to get CTS demand.
- **scenario** (*str*) – Possible options are eGon2035 or eGon100RE

Returns **df_peta_demand** (*pd.DataFrame*) – Annual residential heat demand per building and scenario. Columns of the dataframe are census_population_id and demand.

place_buildings_with_amenities(df, amenities=None, max_amenities=None)

Building centroids are placed randomly within census cells. The Number of buildings is derived from n_amenity_inside, the selected method and number of amenities per building.

Returns **df** (*gpd.GeoDataFrame*) – Table of buildings centroids

remove_double_bus_id(df_cts_buildings)

This is an backup adhoc fix if there should still be a building which is assigned to 2 substations. In this case one of the buildings is just dropped. As this currently accounts for only one building with one amenity the deviation is neglectable.

select_cts_buildings(df_buildings_wo_amenities, max_n)

N Buildings (filtered and synthetic) in each cell with cts demand are selected. Only the first n buildings are taken for each cell. The buildings are sorted by surface area.

Returns **df_buildings_with_cts_demand** (*gpd.GeoDataFrame*) – Table of buildings

hh_buildings

Household electricity demand time series for scenarios in 2035 and 2050 assigned to OSM-buildings.

Assignment of household electricity demand timeseries to OSM buildings and generation of randomly placed synthetic 5x5m buildings if no sufficient OSM-data available in the respective census cell.

The resulting data is stored in separate tables

- **openstreetmap.osm_buildings_synthetic:** Lists generated synthetic building with id and cell_id

- ***demand.egon_household_electricity_profile_of_buildings***: Mapping of demand timeseries and buildings including cell_id, building area and peak load

Both tables are created within `map_houseprofiles_to_buildings()`.

The following datasets from the database are used for creation:

- ***demand.household_electricity_profiles_in_census_cells***: Lists references and scaling parameters to time series data for each household in a cell by identifiers. This table is fundamental for creating subsequent data like demand profiles on MV grid level or for determining the peak load at load. Only the profile reference and the cell identifiers are used.
- ***society.egon_destatis_zensus_apartment_building_population_per_ha***: Lists number of apartments, buildings and population for each census cell.
- ***boundaries.egon_map_zensus_buildings_residential***: List of OSM tagged buildings which are considered to be residential.

What is the goal?

To assign every household demand timeseries, which already exist at cell level, to a specific OSM building.

What is the challenge?

The census and the OSM dataset differ from each other. The census uses statistical methods and therefore lacks accuracy at high spatial resolution. The OSM datasets is community based dataset which is extended throughout and does not claim to be complete. By merging these datasets inconsistencies need to be addressed. For example: not yet tagged buildings in OSM or new building areas not considered in census 2011.

How are these datasets combined?

The assignment of household demand timeseries to buildings takes place at cell level. Within each cell a pool of profiles exists, produced by the ‘HH Demand’ module. These profiles are randomly assigned to a filtered list of OSM buildings within this cell. Every profile is assigned to a building and every building get a profile assigned if there is enough households by the census data. If there are more profiles than buildings, all additional profiles are randomly assigned. Therefore multiple profiles can be assigned to one building, making it a multi-household building.

What are central assumptions during the data processing?

- Mapping zensus data to OSM data is not trivial. Discrepancies are substituted.
- Missing OSM buildings are generated by census building count.
- If no census building count data is available, the number of buildings is

derived by an average rate of households/buildings applied to the number of households.

Drawbacks and limitations of the data

- Missing OSM buildings in cells without census building count are derived by

an average rate of households/buildings applied to the number of households. As only whole houses can exist, the substitute is ceiled to the next higher integer. Ceiling is applied to avoid rounding to amount of 0 buildings.

- As this datasets is a cascade after profile assignement at census cells

also check drawbacks and limitations in `hh_profiles.py`.

Example Query

- Get a list with number of houses, households and household types per census cell

```
SELECT t1.cell_id, building_count, hh_count, hh_types
FROM(
    SELECT cell_id, Count(distinct(building_id)) as building_count,
    count(profile_id) as hh_count
    FROM demand.egon_household_electricity_profile_of_buildings
    Group By cell_id
) as t1
FULL OUTER JOIN(
    SELECT cell_id, array_agg(array[cast(hh_10types as char),
    hh_type]) as hh_types
    FROM society.egon_destatis_zensus_household_per_ha_refined
    GROUP BY cell_id
) as t2
ON t1.cell_id = t2.cell_id
```

Notes

This module docstring is rather a dataset documentation. Once, a decision is made in ... the content of this module docstring needs to be moved to docs attribute of the respective dataset class.

```
class BuildingElectricityPeakLoads(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    building_id
    peak_load_in_w
    scenario
    sector
    voltage_level

class HouseholdElectricityProfilesOfBuildings(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    building_id
    cell_id
    id
    profile_id

class OsmBuildingsSynthetic(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    area
    building
    cell_id
    geom_building
    geom_point
    id
    n_amenities_inside
```

generate_mapping_table (*egon_map_zensus_buildings_residential_synth*,
egon_hh_profile_in_zensus_cell)

Generate a mapping table for hh profiles to buildings.

All hh demand profiles are randomly assigned to buildings within the same census cell.

- **profiles > buildings: buildings can have multiple profiles but every** building gets at least one profile
- profiles < buildings: not every building gets a profile

Parameters

- **egon_map_zensus_buildings_residential_synth** (*pd.DataFrame*) – Table with OSM and synthetic buildings ids per census cell
- **egon_hh_profile_in_zensus_cell** (*pd.DataFrame*) – Table mapping hh demand profiles to census cells

Returns *pd.DataFrame* – Table with mapping of profile ids to buildings with OSM ids

generate_synthetic_buildings (*missing_buildings*, *edge_length*)

Generate synthetic square buildings in census cells for every entry in *missing_buildings*.

Generate random placed synthetic buildings incl geom data within the bounds of the census cell. Buildings have each a square area with edge_length^2 .

Parameters

- **missing_buildings** (*pd.Series* or *pd.DataFrame*) – Table with *cell_ids* and building number
- **edge_length** (*int*) – Edge length of square synthetic building in meter

Returns *pd.DataFrame* – Table with generated synthetic buildings, area, *cell_id* and geom data

get_building_peak_loads ()

Peak loads of buildings are determined.

Timeseries for every building are accumulated, the maximum value determined and with the respective nuts3 factor scaled for 2035 and 2050 scenario.

Note: In test-mode ‘SH’ the iteration takes place by ‘*cell_id*’ to avoid intensive RAM usage. For whole Germany ‘nuts3’ are taken and RAM > 32GB is necessary.

map_houseprofiles_to_buildings ()

Census hh demand profiles are assigned to buildings via osm ids. If no OSM ids available, synthetic buildings are generated. A list of the generated buildings and supplementary data as well as the mapping table is stored in the db.

synthetic_buildings: schema: openstreetmap tablename: osm_buildings_synthetic

mapping_profiles_to_buildings: schema: demand tablename: egon_household_electricity_profile_of_buildings

Notes

match_osm_and_zensus_data (*egon_hh_profile_in_zensus_cell*, *egon_map_zensus_buildings_residential*)

Compares OSM buildings and census hh demand profiles.

OSM building data and hh demand profiles based on census data is compared. Census cells with only profiles but no osm-ids are identified to generate synthetic buildings. Census building count is used, if available, to define

number of missing buildings. Otherwise, the overall mean profile/building rate is used to derive the number of buildings from the number of already generated demand profiles.

Parameters

- **egon_hh_profile_in_zensus_cell** (*pd.DataFrame*) – Table mapping hh demand profiles to census cells
- **egon_map_zensus_buildings_residential** (*pd.DataFrame*) – Table with buildings osm-id and cell_id

Returns *pd.DataFrame* – Table with cell_ids and number of missing buildings

reduce_synthetic_buildings (*mapping_profiles_to_buildings*, *synthetic_buildings*)

Reduced list of synthetic buildings to amount actually used.

Not all are used, due to randomised assignment with replacing Id's are adapted to continuous number sequence following openstreetmap.osm_buildings

hh_profiles

Household electricity demand time series for scenarios in 2035 and 2050 at census cell level.

Electricity demand data for households in Germany in 1-hourly resolution for an entire year. Spatially, the data is resolved to 100 x 100 m cells and provides individual and distinct time series for each household in a cell. The cells are defined by the dataset Zensus 2011.

The resulting data is stored in two separate tables

- *demand.household_electricity_profiles_in_census_cells*: Lists references and scaling parameters to time series data for each household in a cell by identifiers. This table is fundamental for creating subsequent data like demand profiles on MV grid level or for determining the peak load at load area level. The table is created by: `func:houseprofiles_in_census_cells`.
- *demand.household_electricity_profiles_hvmv_substation*: Household electricity demand profiles aggregated at MV grid district level in MWh. Primarily used to create the eTraGo data model. The table is created with `mv_grid_district_HH_electricity_load()`.

The following datasets are used for creating the data:

- Electricity demand time series for household categories produced by demand profile generator (DPG) from Fraunhofer IEE (see `get_iee_hh_demand_profiles_raw()`)
- Spatial information about people living in households by Zensus 2011 at federal state level
 - Type of household (family status)
 - Age
 - Number of people
- Spatial information about number of households per ha, categorized by type of household (family status) with 5 categories (also from Zensus 2011)
- Demand-Regio annual household demand at NUTS3 level

What is the goal?

To use the electricity demand time series from the *demand profile generator* to create spatially reference household demand time series for Germany at a resolution of 100 x 100 m cells.

What is the challenge?

The electricity demand time series produced by demand profile generator offer 12 different household profile categories. To use most of them, the spatial information about the number of households per cell (5 categories) needs to be enriched by supplementary data to match the household demand profile categories specifications. Hence, 10 out of 12 different household profile categories can be distinguished by increasing the number of categories of cell-level household data.

How are these datasets combined?

- Spatial information about people living in households by census (2011) at federal state NUTS1 level **:var:'df_zensus'** is aggregated to be compatible to IEE household profile specifications.
 - exclude kids and reduce to adults and seniors
 - group as defined in **:var:'HH_TYPES'**
 - convert data from people living in households to number of households by **:var:'mapping_people_in_households'**
 - calculate fraction of fine household types (10) within subgroup of rough household types (5) **:var:'df_dist_households'**
- Spatial information about number of households per ha **:var:'df_census_households_nuts3'** is mapped to NUTS1 and NUTS3 level. Data is refined with household subgroups via **:var:'df_dist_households'** to **:var:'df_census_households_grid_refined'**.
- Enriched 100 x 100 m household dataset is used to sample and aggregate household profiles. A table including individual profile id's for each cell and scaling factor to match Demand-Regio annual sum projections for 2035 and 2050 at NUTS3 level is created in the database as *demand.household_electricity_profiles_in_census_cells*.

What are central assumptions during the data processing?

- Mapping census data to IEE household categories is not trivial. In conversion from persons in household to number of households, number of inhabitants for multi-person households is estimated as weighted average in **:var:'OO_factor'**
- The distribution to refine household types at cell level are the same for each federal state
- Refining of household types lead to float number of profiles drew at cell level and need to be rounded to nearest int by `np rint()`.
- 100 x 100 m cells are matched to NUTS via cells centroid location
- Cells with households in unpopulated areas are removed

Drawbacks and limitations of the data

- The distribution to refine household types at cell level are the same for each federal state
- Household profiles aggregated annual demand matches Demand Regio demand at NUTS-3 level, but it is not matching the demand regio time series profile
- Due to secrecy, some census data are highly modified under certain attributes (quantity_q = 2). This cell data is not corrected, but excluded.
- There is deviation in the Census data from table to table. The statistical methods are not stringent. Hence, there are cases in which data contradicts.
- Census data with attribute 'HHTYP_FAM' is missing for some cells with small amount of households. This data is generated using the average share of household types for cells with similar household number. For some cells the summed amount of households per type deviates from the total number with attribute 'INSGESAMT'. As the profiles are scaled with demand-regio data at nuts3-level the impact at a higher aggregation level is negligible. For sake of simplicity, the data is not corrected.

- There are cells without household data but a population. A randomly chosen household distribution is taken from a subgroup of cells with same population value and applied to all cells with missing household distribution and the specific population value.

Helper functions

- To access the DB, select specific profiles at various aggregation levels

use:func:*get_hh_profiles_from_db* * To access the DB, select specific profiles at various aggregation levels and scale profiles use :func:*get_scaled_profiles_from_db*

Notes

This module docstring is rather a dataset documentation. Once, a decision is made in ... the content of this module docstring needs to be moved to docs attribute of the respective dataset class.

```
class EgonDestatisZensusHouseholdPerHaRefined (**kwargs)
```

```
    Bases: sqlalchemy.ext.declarative.api.Base
```

```
    cell_id
```

```
    characteristics_code
```

```
    grid_id
```

```
    hh_10types
```

```
    hh_5types
```

```
    hh_type
```

```
    id
```

```
    nuts1
```

```
    nuts3
```

```
class EgonEtragoElectricityHouseholds (**kwargs)
```

```
    Bases: sqlalchemy.ext.declarative.api.Base
```

```
    bus_id
```

```
    p_set
```

```
    q_set
```

```
    scn_name
```

```
class HouseholdDemands (dependencies)
```

```
    Bases: egon.data.datasets.Dataset
```

```
class HouseholdElectricityProfilesInCensusCells (**kwargs)
```

```
    Bases: sqlalchemy.ext.declarative.api.Base
```

```
    cell_id
```

```
    cell_profile_ids
```

```
    factor_2035
```

```
    factor_2050
```

```
    grid_id
```

```

nuts1
nuts3
class IeeHouseholdLoadProfiles (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    id
    load_in_wh
    type
adjust_to_demand_regio_nuts3_annual (df_hh_profiles_in_census_cells, df_iee_profiles,
                                     df_demand_regio)
    Computes the profile scaling factor for alignment to demand regio data

    The scaling factor can be used to re-scale each load profile such that the sum of all load profiles within one
    NUTS-3 area equals the annual demand of demand regio data.

```

Parameters

- **df_hh_profiles_in_census_cells** (*pd.DataFrame*) – Result of [assign_hh_demand_profiles_to_cells\(\)](#).
- **df_iee_profiles** (*pd.DataFrame*) – Household load profile data
 - Index: Times steps as serial integers
 - Columns: *pd.MultiIndex* with (*HH_TYPE*, *id*)
- **df_demand_regio** (*pd.DataFrame*) – Annual demand by demand regio for each NUTS-3 region and scenario year. Index is *pd.MultiIndex* with tuple(*scenario_year*, *nuts3_code*).

Returns *pd.DataFrame* – Returns the same data as [assign_hh_demand_profiles_to_cells\(\)](#), but with filled columns *factor_2035* and *factor_2050*.

```

assign_hh_demand_profiles_to_cells (df_zensus_cells, df_iee_profiles)

```

Assign household demand profiles to each census cell.

A table including the demand profile ids for each cell is created by using [get_cell_demand_profile_ids\(\)](#). Household profiles are randomly sampled for each cell. The profiles are not replaced to the pool within a cell but after.

Parameters

- **df_zensus_cells** (*pd.DataFrame*) – Household type parameters. Each row representing one household. Hence, multiple rows per zensus cell.
- **df_iee_profiles** (*pd.DataFrame*) – Household load profile data
 - Index: Times steps as serial integers
 - Columns: *pd.MultiIndex* with (*HH_TYPE*, *id*)

Returns *pd.DataFrame* – Tabular data with one row represents one zensus cell. The column *cell_profile_ids* contains a list of tuples (see [get_cell_demand_profile_ids\(\)](#)) providing a reference to the actual load profiles that are associated with this cell.

```

clean (x)

```

Clean zensus household data row-wise

Clean dataset by

- converting ‘.’ and ‘-’ to str(0)

- removing brackets

Table can be converted to int/floats afterwards

Parameters *x* (*pd.Series*) – It is meant to be used with `df.applymap()`

Returns *pd.Series* – Re-formatted data row

create_missing_zensus_data (*df_households_typ*, *df_missing_data*, *missing_cells*)

There is missing data for specific attributes in the zensus dataset because of secrecy reasons. Some cells with only small amount of households are missing with attribute HHTYP_FAM. However the total amount of households is known with attribute INSGESAMT. The missing data is generated as average share of the household types for cell groups with the same amount of households.

Parameters

- **df_households_typ** (*pd.DataFrame*) – Zensus households data
- **df_missing_data** (*pd.DataFrame*) – number of missing cells of group of amount of households
- **missing_cells** (*dict*) – dictionary with list of grids of the missing cells grouped by amount of households in cell

Returns *df_average_split* (*pd.DataFrame*) – generated dataset of missing cells

get_cell_demand_metadata_from_db (*attribute*, *list_of_identifiers*)

Retrieve selection of household electricity demand profile mapping

Parameters

- **attribute** (*str*) – attribute to filter the table
 - nuts3
 - nuts1
 - cell_id
- **list_of_identifiers** (*list of str/int*) – nuts3/nuts1 need to be str cell_id need to be int

See also:

`houseprofiles_in_census_cells()`

Returns *pd.DataFrame* – Selection of mapping of household demand profiles to zensus cells

get_cell_demand_profile_ids (*df_cell*, *pool_size*)

Generates tuple of hh_type and zensus cell ids

Takes a random sample of profile ids for given cell:

- if pool size \geq sample size: without replacement
- if pool size $<$ sample size: with replacement

Parameters

- **df_cell** (*pd.DataFrame*) – Household type information for a single zensus cell
- **pool_size** (*int*) – Number of available profiles to select from

Returns *list of tuple* – List of (*hh_type*, *cell_id*)

get_census_households_grid()

Query census household data at 100x100m grid level from database. As there is a divergence in the census household data depending which attribute is used. There also exist cells without household but with population data. The missing data in these cases are substituted. First census household data with attribute 'HHTYP_FAM' is missing for some cells with small amount of households. This data is generated using the average share of household types for cells with similar household number. For some cells the summed amount of households per type deviates from the total number with attribute 'INSGESAMT'. As the profiles are scaled with demand-regio data at nuts3-level the impact at a higher aggregation level is negligible. For sake of simplicity, the data is not corrected.

Returns *pd.DataFrame* – census household data at 100x100m grid level

get_census_households_nuts1_raw()

Get zensus age x household type data from egon-data-bundle

Dataset about household size with information about the categories:

- family type
- age class
- household size

for Germany in spatial resolution of federal states NUTS-1.

Data manually selected and retrieved from: <https://ergebnisse2011.zensus2022.de/datenbank/online> For reproducing data selection, please do:

- Search for: “1000A-3016”
- or choose topic: “Bevölkerung kompakt”
- Choose table code: “1000A-3016” with title “Personen: Alter (11 Altersklassen) - Größe des privaten Haushalts - Typ des privaten Haushalts (nach Familien/Lebensform)”
- Change setting “GEOLK1” to “Bundesländer (16)”

Data would be available in higher resolution (“Landkreise und kreisfreie Städte (412)”), but only after registration.

The downloaded file is called ‘Zensus2011_Personen.csv’.

Returns *pd.DataFrame* – Pre-processed zensus household data

get_hh_profiles_from_db(profile_ids)

Retrieve selection of household electricity demand profiles

Parameters *profile_ids* (list of str (str, int)) – (type)a00..(profile number) with number having exactly 4 digits

See also:

houseprofiles_in_census_cells()

Returns *pd.DataFrame* – Selection of household demand profiles

get_houseprofiles_in_census_cells()

Retrieve household electricity demand profile mapping from database

See also:

houseprofiles_in_census_cells()

Returns *pd.DataFrame* – Mapping of household demand profiles to census cells

get_iee_hh_demand_profiles_raw()

Gets and returns household electricity demand profiles from the egon-data-bundle.

Household electricity demand profiles generated by Fraunhofer IEE. Methodology is described in [:ref:‘Erzeugung zeitlich hochaufgelöster Stromlastprofile für verschiedene Haushaltstypen](https://www.researchgate.net/publication/273775902_Erzeugung_zeitlich_hochaufgeloster_Stromlastprofile_fur_verschiedene_Haushaltstypen)

[<https://www.researchgate.net/publication/273775902_Erzeugung_zeitlich_hochaufgeloster_Stromlastprofile_fur_verschiedene_Haushaltstypen](https://www.researchgate.net/publication/273775902_Erzeugung_zeitlich_hochaufgeloster_Stromlastprofile_fur_verschiedene_Haushaltstypen)

It is used and further described in the following theses by:

- Jonas Haack: “Auswirkungen verschiedener Haushaltslastprofile auf PV-Batterie-Systeme” (confidential)
- Simon Ruben Drauz “Synthesis of a heat and electrical load profile for single and multi-family houses used for subsequent performance tests of a multi-component energy system”, <http://dx.doi.org/10.13140/RG.2.2.13959.14248>

Notes

The household electricity demand profiles have been generated for 2016 which is a leap year (8784 hours) starting on a Friday. The weather year is 2011 and the heat timeseries 2011 are generated for 2011 too (cf. dataset `egon.data.datasets.heat_demand_timeseries.HTS`), having 8760h and starting on a Saturday. To align the profiles, the first day of the IEE profiles are deleted, resulting in 8760h starting on Saturday.

Returns *pd.DataFrame* – Table with profiles in columns and time as index. A *pd.MultiIndex* is used to distinguish load profiles from different EUROSTAT household types.

get_load_timeseries (*df_iee_profiles*, *df_hh_profiles_in_census_cells*, *cell_ids*, *year*, *aggregate=True*, *peak_load_only=False*)

Get peak load for one load area in MWh

The peak load is calculated in aggregated manner for a group of census cells that belong to one load area (defined by *cell_ids*).

Parameters

- **df_iee_profiles** (*pd.DataFrame*) – Household load profile data in Wh
 - Index: Times steps as serial integers
 - Columns: *pd.MultiIndex* with (*HH_TYPE*, *id*)Used to calculate the peak load from.
- **df_hh_profiles_in_census_cells** (*pd.DataFrame*) – Return value of `adjust_to_demand_regio_nuts3_annual()`.
- **cell_ids** (*list*) – Census cell ids that define one group of census cells that belong to the same load area.
- **year** (*int*) – Scenario year. Is used to consider the scaling factor for aligning annual demand to NUTS-3 data.
- **aggregate** (*bool*) – If true, all profiles are aggregated
- **peak_load_only** (*bool*) – If true, only the peak load value is returned (the type of the return value is *float*). Defaults to False which returns the entire time series as *pd.Series*.

Returns *pd.Series* or *float* – Aggregated time series for given *cell_ids* or peak load of this time series in MWh.

get_scaled_profiles_from_db (*attribute*, *list_of_identifiers*, *year*, *aggregate=True*, *peak_load_only=False*)

Retrieve selection of scaled household electricity demand profiles

Parameters

- **attribute** (*str*) – attribute to filter the table
 - nuts3
 - nuts1
 - cell_id
- **list_of_identifiers** (*list of str/int*) – nuts3/nuts1 need to be str cell_id need to be int
- **year** (*int*) –
 - 2035
 - 2050
- **aggregate** (*bool*) – If True, all profiles are summed. This uses a lot of RAM if a high attribute level is chosen
- **peak_load_only** (*bool*) – If True, only peak load value is returned

Notes

Aggregate == False option can use a lot of RAM if many profiles are selected

Returns *pd.Series or float* – Aggregated time series for given *cell_ids* or peak load of this time series in MWh.

houseprofiles_in_census_cells ()

Allocate household electricity demand profiles for each census cell.

Creates a table that maps household electricity demand profiles to census cells. Each row represents one cell and contains a list of profile IDs.

Use `get_houseprofiles_in_census_cells()` to retrieve the data from the database as pandas

impute_missing_hh_in_populated_cells (*df_census_households_grid*)

There are cells without household data but a population. A randomly chosen household distribution is taken from a subgroup of cells with same population value and applied to all cells with missing household distribution and the specific population value. In the case, in which there is no subgroup with household data of the respective population value, the fallback is the subgroup with the last last smaller population value.

Parameters *df_census_households_grid* (*pd.DataFrame*) – census household data at 100x100m grid level

Returns *pd.DataFrame* – substituted census household data at 100x100m grid level

inhabitants_to_households (*df_hh_people_distribution_abs*)

Convert number of inhabitant to number of household types

Takes the distribution of peoples living in types of households to calculate a distribution of household types by using a people-in-household mapping. Results are not rounded to int as it will be used to calculate a relative distribution anyways. The data of category 'HHGROESS_KLASS' in census households at grid level is used to determine an average wherever the amount of people is not trivial (OR, OO). Kids are not counted.

Parameters *df_hh_people_distribution_abs* (*pd.DataFrame*) – Grouped census household data on NUTS-1 level in absolute values

Returns *df_dist_households* (*pd.DataFrame*) – Distribution of households type

mv_grid_district_HH_electricity_load (*scenario_name*, *scenario_year*, *drop_table=False*)

Aggregated household demand time series at HV/MV substation level

Calculate the aggregated demand time series based on the demand profiles of each census cell inside each MV grid district. Profiles are read from local hdf5-file.

Parameters

- **scenario_name** (*str*) – Scenario name identifier, i.e. “eGon2035”
- **scenario_year** (*int*) – Scenario year according to *scenario_name*
- **drop_table** (*bool*) – Toggle to True for dropping table at beginning of this function. Be careful, delete any data.

Returns *pd.DataFrame* – Multiindexed dataframe with *timestep* and *bus_id* as indexers. Demand is given in kWh.

process_nuts1_census_data (*df_census_households_raw*)

Make data compatible with household demand profile categories

Removes and reorders categories which are not needed to fit data to household types of IEE electricity demand time series generated by demand-profile-generator (DPG).

- Kids (<15) are excluded as they are also excluded in DPG origin dataset
- Adults (15<65)
- Seniors (<65)

Parameters *df_census_households_raw* (*pd.DataFrame*) – cleaned census household type x age category data

Returns *pd.DataFrame* – Aggregated census household data on NUTS-1 level

proportionate_allocation (*df_group*, *dist_households_nuts1*, *hh_10types_cluster*)

Household distribution at nuts1 are applied at census cell within group

To refine the hh_5types and keep the distribution at nuts1 level, the household types are clustered and drawn with proportionate weighting. The resulting pool is splitted into subgroups with sizes according to the number of households of clusters in cells.

Parameters

- **df_group** (*pd.DataFrame*) – Census household data at grid level for specific hh_5type cluster in a federal state
- **dist_households_nuts1** (*pd.Series*) – Household distribution of of hh_10types in a federal state
- **hh_10types_cluster** (*list of str*) – Cluster of household types to be refined to

Returns *pd.DataFrame* – Refined household data with hh_10types of cluster at nuts1 level

refine_census_data_at_cell_level (*df_census_households_grid*, *df_census_households_nuts1*)

The census data is processed to define the number and type of households per census cell. Two subsets of the census data are merged to fit the IEE profiles specifications. To do this, proportionate allocation is applied at nuts1 level and within household type clusters.

Header “characteristics_code”, “characteristics_text”, “mapping”

“1”, “Einpersonenhaushalte (Singlehaushalte)”, “SR; SO” “2”, “Paare ohne Kind(er)”, “PR; PO” “3”, “Paare mit Kind(ern)”, “P1; P2; P3” “4”, “Alleinerziehende Elternteile”, “SK” “5”, “Mehrpersonenhaushalte ohne Kernfamilie”, “OR; OO”

Parameters

- **df_census_households_grid** (*pd.DataFrame*) – Aggregated census household data on 100x100m grid level
- **df_census_households_nuts1** (*pd.DataFrame*) – Aggregated census household data on NUTS-1 level

Returns *pd.DataFrame* – Number of hh types per census cell

regroup_nuts1_census_data (*df_census_households_nuts1*)

Regroup census data and map according to demand-profile types. For more information look at the respective publication: https://www.researchgate.net/publication/273775902_Erzeugung_zeitlich_hochaufgeloster_Stromlastprofile_fur_verschiedene_Haushaltstypen

Parameters **df_census_households_nuts1** (*pd.DataFrame*) – census household data on NUTS-1 level in absolute values

Returns **df_dist_households** (*pd.DataFrame*) – Distribution of households type

set_multiindex_to_profiles (*hh_profiles*)

The profile id is split into type and number and set as multiindex.

Parameters **hh_profiles** (*pd.DataFrame*) – Profiles

Returns **hh_profiles** (*pd.DataFrame*) – Profiles with Multiindex

write_hh_profiles_to_db (*hh_profiles*)

Write HH demand profiles of IEE into db. One row per profile type. The annual load profile timeseries is an array.

schema: demand tablename: iee_household_load_profiles

Parameters **hh_profiles** (*pd.DataFrame*) – It is meant to be used with `df.applymap()`

write_refinded_households_to_db (*df_census_households_grid_refined*)

mapping

class EgonMapZensusMvgdBBuildings (***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`

building_id

bus_id

electricity

heat

osm

sector

zensus_population_id

map_all_used_buildings ()

This function maps all used buildings from OSM and synthetic ones.

tools

psql_insert_copy (*table, conn, keys, data_iter*)

Execute SQL statement inserting data

Parameters

- **table** (*pandas.io.sql.SQLiteTable*)
- **conn** (*sqlalchemy.engine.Engine* or *sqlalchemy.engine.Connection*)
- **keys** (*list of str*) – Column names
- **data_iter** (*Iterable that iterates the values to be inserted*)

random_ints_until_sum (*s_sum, m_max*)

Generate non-negative random integers < *m_max* summing to *s_sum*.

random_point_in_square (*geom, tol*)

Generate a random point within a square

Parameters

- **geom** (*gpd.Series*) – Geometries of square
- **tol** (*float*) – tolerance to square bounds

Returns *points* (*gpd.Series*) – Series of random points

specific_int_until_sum (*s_sum, i_int*)

Generate list *i_int* summing to *s_sum*. Last value will be <= *i_int*

timeit (*func*)

Decorator for measuring function's running time.

write_table_to_postgis (*gdf, table, engine=Engine(postgresql+psycopg2://egon:***@127.0.0.1:59734/egon-data), drop=True*)

Helper function to append df data to table in db. Only predefined columns are passed. Error will raise if column is missing. Dtype of columns are taken from table definition.

Parameters

- **gdf** (*gpd.DataFrame*) – Table of data
- **table** (*declarative_base*) – Metadata of db table to export to
- **engine** – connection to database db.engine()
- **drop** (*bool*) – Drop table before appending

write_table_to_postgres (*df, db_table, drop=False, index=False, if_exists='append'*)

Helper function to append df data to table in db. Fast string-copy is used. Only predefined columns are passed. If column is missing in dataframe a warning is logged. Dtypes of columns are taken from table definition. The writing process happens in a scoped session.

Parameters

- **df** (*pd.DataFrame*) – Table of data
- **db_table** (*declarative_base*) – Metadata of db table to export to
- **drop** (*boolean, default False*) – Drop db-table before appending
- **index** (*boolean, default False*) – Write DataFrame index as a column.
- **if_exists** (*{'fail', 'replace', 'append'}, default 'append'*) –
 - fail: If table exists, do nothing.
 - replace: If table exists, drop it, recreate it, and insert data.
 - append: If table exists, insert data. Create if does not exist.

10.5.36 gas_neighbours

eGon100RE

Module containing code dealing with cross border gas pipelines for eGon100RE

In this module the cross border pipelines for H2 and CH4, exclusively between Germany and its neighbouring countries, in eGon100RE are defined and inserted in the database.

Dependencies (pipeline)

- **dataset** PypsaEurSec, GasNodesandPipes, HydrogenBusEtrago, ElectricalNeighbours

Resulting tables

- grid.egon_etrigo_link is completed

calculate_crossbordering_gas_grid_capacities_eGon100RE (*cap_DE*,
DE_pipe_capacities_list)

Attribute gas cross border grid capacities for eGon100RE

This function attributes to each cross border pipeline (H2 and CH4) between Germany and its neighbouring countries its capacity.

Parameters

- **cap_DE** (*pandas.DataFrame*) – List of the H2 and CH4 exchange capacity for each neighbouring country of Germany.
- **DE_pipe_capacities_list** (*pandas.DataFrame*) – List of the cross border for H2 and CH4 pipelines between Germany and its neighbouring countries in eGon100RE, with geometry (geom and topo) but no capacity.

Returns **Crossbordering_pipe_capacities_list** (*pandas.DataFrame*) – List of the cross border H2 and CH4 pipelines between Germany and its neighbouring countries in eGon100RE.

define_DE_crossbording_pipes_geom_eGon100RE (*scn_name='eGon100RE'*)

Define the missing cross border gas pipelines in eGon100RE

This function defines the cross border pipelines (for H2 and CH4) between Germany and its neighbouring countries. These pipelines are defined as links and there are copied from the corresponding CH4 cross border pipelines from eGon2035.

Parameters **scn_name** (*str*) – Name of the scenario

Returns **gas_pipelines_list_DE** (*pandas.DataFrame*) – List of the cross border H2 and CH4 pipelines between Germany and its neighbouring countries in eGon100RE, with geometry (geom and topo) but no capacity.

insert_gas_neighbours_eGon100RE ()

Insert missing gas cross border grid capacities for eGon100RE

This function insert the cross border pipelines for H2 and CH4, exclusively between Germany and its neighbouring countries, for eGon100RE in the database by executing the following steps:

- call of the function `define_DE_crossbording_pipes_geom_eGon100RE()`, that defines the cross border pipelines (H2 and CH4) between Germany and its neighbouring countries

- call of the function `read_DE_crossbordering_cap_from_pes()`, that calculates the cross border total exchange capacities for H2 and CH4 between Germany and its neighbouring countries based on the pypsa-eur-sec results
- call of the function `calculate_crossbordering_gas_grid_capacities_eGon100RE()`, that attributes to each cross border pipeline (H2 and CH4) between Germany and its neighbouring countries its capacity
- insertion of the H2 and CH4 pipelines between Germany and its neighbouring countries in the database with function `insert_gas_grid_capacities()`

Returns *None*

`read_DE_crossbordering_cap_from_pes()`

Read gas pipelines cross border capacities from pes run

This function calculates the cross border total exchange capacities for H2 and CH4 between Germany and its neighbouring countries based on the pypsa-eur-sec results.

Returns `DE_pipe_capacities_list` (*pandas.DataFrame*) – List of the H2 and CH4 exchange capacity for each neighbouring country of Germany.

eGon2035

Central module containing code dealing with gas neighbours for eGon2035

`calc_capacities()`

Calculates gas production capacities of neighbouring countries

For each neighbouring country, this function calculates the gas generation capacity in 2035 using the function `calc_capacity_per_year()` for 2030 and 2040 and interpolates the results. These capacities include LNG import, as well as conventional and biogas production. Two conventional gas generators are added for Norway and Russia interpolating the supply potential values from the TYNPD 2020 for 2030 and 2040.

Returns `grouped_capacities` (*pandas.DataFrame*) – Gas production capacities per foreign node

`calc_capacity_per_year(df, lng, year)`

Calculates gas production capacities for a specified year

For a specified year and for the foreign country nodes this function calculates the gas production capacities, considering the gas (conventional and bio) production capacities from TYNDP data and the LNG import capacities from Scigrid gas data.

The columns of the returned dataframe are the following:

- `Value_bio_year`: biogas production capacity (in GWh/d)
- `Value_conv_year`: conventional gas production capacity including LNG imports (in GWh/d)
- `CH4_year`: total gas production capacity (in GWh/d). This value is calculated using the peak production value from the TYNDP.
- `e_nom_max_year`: total gas production capacity representative for the whole year (in GWh/d). This value is calculated using the average production value from the TYNDP and will then be used to limit the energy that can be generated in one year.
- `share_LNG_year`: share of LGN import capacity in the total gas production capacity
- `share_conv_pipe_year`: share of conventional gas extraction capacity in the total gas production capacity
- `share_bio_year`: share of biogas production capacity in the total gas production capacity

Parameters

- **df** (*pandas.DataFrame*) – Gas (conventional and bio) production capacities from TYNDP (in GWh/d)
- **lng** (*pandas.Series*) – LNG terminal capacities per foreign country node (in GWh/d)
- **year** (*int*) – Year to calculate gas production capacities for

Returns **df_year** (*pandas.DataFrame*) – Gas production capacities (in GWh/d) per foreign country node

calc_ch4_storage_capacities()

Calculate CH4 storage capacities for neighboring countries

Returns

- **ch4_storage_capacities** (*pandas.DataFrame*)
- *Methane gas storage capacities per country in MWh*

calc_global_ch4_demand(Norway_global_demand_ly)

Calculates global CH4 demands abroad for eGon2035 scenario

The data comes from TYNDP 2020 according to NEP 2021 from the scenario ‘Distributed Energy’; linear interpolates between 2030 and 2040.

Returns *pandas.DataFrame* – Global (yearly) CH4 final demand per foreign node

calc_global_power_to_h2_demand()

Calculate H2 demand abroad for eGon2035 scenario

Calculates global power demand abroad linked to H2 production. The data comes from TYNDP 2020 according to NEP 2021 from the scenario ‘Distributed Energy’; linear interpolate between 2030 and 2040.

Returns **global_power_to_h2_demand** (*pandas.DataFrame*) – Global hourly power-to-h2 demand per foreign node

calculate_ch4_grid_capacities()

Calculates CH4 grid capacities for foreign countries based on TYNDP-data

Returns **Neighbouring_pipe_capacities_list** (*pandas.DataFrame*) – Table containing the CH4 grid capacity for each foreign country

calculate_ocgt_capacities()

Calculate gas turbine capacities abroad for eGon2035

Calculate gas turbine capacities abroad for eGon2035 based on TYNDP 2020, scenario “Distributed Energy”; interpolated between 2030 and 2040

Returns **df_ocgt** (*pandas.DataFrame*) – Gas turbine capacities per foreign node

get_foreign_gas_bus_id(carrier='CH4')

Calculate the etrago bus id based on the geometry

Map node_ids from TYNDP and etragos bus_id

Parameters **carrier** (*str*) – Name of the carrier

Returns *pandas.Series* – List of mapped node_ids from TYNDP and etragos bus_id

grid()

Insert data from TYNDP 2020 according to NEP 2021 Scenario ‘Distributed Energy’; linear interpolate between 2030 and 2040

Returns *None*

import_ch4_demandTS ()

Calculate global CH4 demand in Norway and CH4 demand profile

Import from the PyPSA-eur-sec run the time series of residential rural heat per neighbor country. This time series is used to calculate:

- the global (yearly) heat demand of Norway (that will be supplied by CH4)
- the normalized CH4 hourly resolved demand profile

Returns

- **Norway_global_demand** (*Float*) – Yearly heat demand of Norway in MWh
- **neighbor_loads_t** (*pandas.DataFrame*) – Normalized CH4 hourly resolved demand profiles per neighbor country

insert_ch4_demand (global_demand, normalized_ch4_demandTS)

Insert CH4 demands abroad into the database for eGon2035

Parameters

- **global_demand** (*pandas.DataFrame*) – Global CH4 demand per foreign node in 1 year
- **gas_demandTS** (*pandas.DataFrame*) – Normalized time series of the demand per foreign country

Returns *None***insert_generators (gen)**

Insert gas generators for foreign countries into the database

Insert gas generators for foreign countries into the database. The marginal cost of the methane is calculated as the sum of the imported LNG cost, the conventional natural gas cost and the biomethane cost, weighted by their share in the total import/ production capacity. LNG gas is considered to be 30% more expensive than the natural gas transported by pipelines (source: iwd, 2022).

Parameters **gen** (*pandas.DataFrame*) – Gas production capacities per foreign node and energy carrier

Returns *None***insert_ocgt_abroad ()**

Insert gas turbine capacities abroad for eGon2035 in the database

Parameters **df_ocgt** (*pandas.DataFrame*) – Gas turbine capacities per foreign node

Returns *None***insert_power_to_h2_demand (global_power_to_h2_demand)**

Insert H2 demands into database for eGon2035

Parameters **global_power_to_h2_demand** (*pandas.DataFrame*) – Global hourly power-to-h2 demand per foreign node

Returns *None***insert_storage (ch4_storage_capacities)**

Insert CH4 storage capacities into the database for eGon2035

Parameters **ch4_storage_capacities** (*pandas.DataFrame*) – Methane gas storage capacities per country in MWh

Returns *None*

read_LNG_capacities()

Read LNG import capacities from Scigrid gas data

Returns `IGGIELGN_LNGs` (*pandas.Series*) – LNG terminal capacities per foreign country node (in GWh/d)

tyndp_gas_demand()

Insert gas demands abroad for eGon2035

Insert CH4 and H2 demands abroad for eGon2035 by executing the following steps:

- **CH4**

- Calculation of the global CH4 demand in Norway and the CH4 demand profile by executing the function `import_ch4_demandTS()`
- Calculation of the global CH4 demands by executing the function `calc_global_ch4_demand()`
- Insertion of the CH4 loads and their associated time series in the database by executing the function `insert_ch4_demand()`

- **H2**

- Calculation of the global power demand abroad linked to H2 production by executing the function `calc_global_power_to_h2_demand()`
- Insertion of these loads in the database by executing the function `insert_power_to_h2_demand()`

Returns *None*

tyndp_gas_generation()

Insert data from TYNDP 2020 according to NEP 2021 Scenario ‘Distributed Energy’; linear interpolate between 2030 and 2040

Returns *None*

gas_abroad

Module containing functions to insert gas abroad

In this module, functions used to insert the gas components (H2 and CH4) abroad for eGon2035 and eGon100RE are defined.

insert_gas_grid_capacities (*Neighbouring_pipe_capacities_list*, *scn_name*)

Insert crossbordering gas pipelines into the database

This function inserts a list of crossbordering gas pipelines after cleaning the database. For eGon2035, all the CH4 crossbordering pipelines are inserted (no H2 grid in this scenario). For eGon100RE, only the crossbordering pipelines with Germany are inserted (the other ones are inserted in PypsaEurSec), but in this scenario there are H2 and CH4 pipelines.

Parameters

- **Neighbouring_pipe_capacities_list** (*pandas.DataFrame*) – List of the crossbordering gas pipelines
- **scn_name** (*str*) – Name of the scenario

Returns *None*

The central module containing all code dealing with gas neighbours

```
class GasNeighbours (dependencies)
    Bases: egon.data.datasets.Dataset
```

10.5.37 heat_demand

Central module containing all code dealing with the future heat demand import.

This module obtains the residential and service-sector heat demand data for 2015 from Peta5.0.1, calculates future heat demands and saves them in the database with assigned census cell IDs.

```
class EgonPetaHeat (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    demand
    id
    scenario
    sector
    zensus_population_id
```

```
class HeatDemandImport (dependencies)
    Bases: egon.data.datasets.Dataset
```

Insert the annual heat demand per census cell for each scenario

This dataset downloads the heat demand raster data for private households and CTS from Peta 5.0.1 (<https://s-energies-open-data-euf.hub.arcgis.com/maps/d7d18b63250240a49eb81db972aa573e/about>) and stores it into files in the working directory. The data from Peta 5.0.1 represents the status quo of the year 2015. To model future heat demands, the data is scaled to meet target values from external sources. These target values are defined for each scenario in *ScenarioParameters*.

Dependencies

- *ScenarioParameters*
- *Vg250*
- *ZensusVg250*

Resulting tables

- *demand.egon_peta_heat* is created and filled

```
name = 'heat-demands'
```

```
version = '0.0.1'
```

```
add_metadata()
    Writes metadata JSON string into table comment.
```

```
adjust_residential_heat_to_zensus (scenario)
    Adjust residential heat demands to fit to zensus population.
```

In some cases, Peta assigns residential heat demand to unpopulated cells. This can be caused by the different population data used in Peta or buildings in zensus cells without a population (see *egon.data.importing.zensus.adjust_zensus_misc()*)

Residential heat demand in cells without zensus population is dropped. Residential heat demand in cells with zensus population is scaled to meet the overall residential heat demands.

Parameters *scenario* (*str*) – Name of the scenario.

Returns *None*

cutout_heat_demand_germany ()

Save cutouts of Germany's 2015 heat demand densities from Europe-wide tifs.

1. Get the German state boundaries
2. Load the unzip 2015 heat demand data (Peta5_0_1) and
3. Cutout Germany's residential and service-sector heat demand densities
4. Save the cutouts as tiffs

Parameters *None*

Returns *None*

Notes

The alternative of cutting out Germany from the pan-European raster based on German census cells, instead of using state boundaries with low resolution (to avoid inaccuracies), was not implemented in order to achieve consistency with other datasets (e.g. `egon_mv_grid_district`). Besides, all attempts to read, (union) and load cells from the local database failed, but were documented as commented code within this function and afterwards removed. If you want to have a look at the comments, please check out commit `ec3391e182215b32cd8b741557a747118ab61664`, which is the last commit still containing them.

Also the usage of a buffer around the boundaries and the subsequent selection of German cells was not implemented. could be used, but then it must be ensured that later only heat demands of cells belonging to Germany are used.

download_peta5_0_1_heat_demands ()

Download Peta5.0.1 tiff files.

The downloaded data contain residential and service-sector heat demands per hectare grid cell for 2015.

Parameters *None*

Returns *None*

Notes

The heat demand data in the Peta5.0.1 dataset are assumed not change. An upgrade to a higher Peta version is currently not foreseen. Therefore, for the version management we can assume that the dataset will not change, unless the code is changed.

future_heat_demand_germany (*scenario_name*)

Calculate the future residential and service-sector heat demand per ha.

The calculation is based on Peta5_0_1 heat demand densities, cutout for Germany, for the year 2015. The given scenario name is used to read the adjustment factors for the heat demand rasters from the scenario table.

Parameters **scenario_name** (*str*) – Selected scenario name for which assumptions will be loaded.

Returns *None*

Notes

None

heat_demand_to_db_table()

Import heat demand rasters and convert them to vector data.

Specify the rasters to import as raster file patterns (file type and directory containing raster files, which all will be imported). The rasters are stored in a temporary table called “heat_demand_rasters”. The final demand data, having the census IDs as foreign key (from the census population table), are generated by the provided sql script (raster2cells-and-centroids.sql) and are stored in the table “demand.egon_peta_heat”.

Parameters None

Returns *None*

Notes

Please note that the data from “demand.egon_peta_heat” is deleted prior to the import, so make sure you’re not losing valuable data.

scenario_data_import()

Call all heat demand import related functions.

This function executes the functions that download, unzip and adjust the heat demand distributions from Peta5.0.1 and that save the future heat demand distributions for Germany as tiffs as well as with census grid IDs as foreign key in the database.

Parameters None

Returns *None*

Notes

None

unzip_peta5_0_1_heat_demands()

Unzip the downloaded Peta5.0.1 tiff files.

Parameters None

Returns *None*

Notes

It is assumed that the Peta5.0.1 dataset does not change and that the version number does not need to be checked.

10.5.38 heat_demand_timeseries

daily

class EgonDailyHeatDemandPerClimateZone (**kwargs)

Bases: sqlalchemy.ext.declarative.api.Base

climate_zone

daily_demand_share

day_of_year

temperature_class

```

class EgonMapZensusClimateZones (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    climate_zone

    zensus_population_id

class IdpProfiles (df_index, **kwargs)
    Bases: object

    get_temperature_interval (how='geometric_series')
        Appoints the corresponding temperature interval to each temperature in the temperature vector.

daily_demand_shares_per_climate_zone ()
    Calculates shares of heat demand per day for each climate zone

    Returns None.

h_value ()
    Description: Assignment of daily demand scaling factor to each day of all TRY Climate Zones

    Returns h (pandas.DataFrame) – Hourly factor values for each station corresponding to the temperature profile. Extracted from demandlib.

map_climate_zones_to_zensus ()
    Geospatial join of zensus cells and climate zones

    Returns None.

temp_interval ()
    Description: Create Dataframe with temperature data for TRY Climate Zones :returns: temperature_interval (pandas.DataFrame) – Hourly temperature interval of all 15 TRY Climate station#s temperature profile

temperature_classes ()

temperature_profile_extract ()
    Description: Extract temperature data from atlite :returns: temperature_profile (pandas.DataFrame) – Temperature profile of all TRY Climate Zones 2011

```

idp_pool

```

class EgonHeatTimeseries (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    building_id

    selected_idp_profiles

    zensus_population_id

annual_demand_generator ()
    Description: Create dataframe with annual demand and household count for each zensus cell :returns: demand_count (pandas.DataFrame) – Annual demand of all zensus cell with MFH and SFH count and respective associated Station

create ()
    Description: Create dataframe with all temperature classes, 24hr. profiles and household stock

    Returns idp_df (pandas.DataFrame) – All IDP pool as classified as per household stock and temperature class

idp_pool_generator ()
    Description: Create List of Dataframes for each temperature class for each household stock

```

TYPE list List of dataframes with each element representing a dataframe for every combination of household stock and temperature class

select ()

Random assignment of intray-day profiles to each day based on their temperature class and household stock count

Returns *None*.

temperature_classes ()

service_sector

CTS_demand_scale (*aggregation_level*)

Description: caling the demand curves to the annual demand of the respective aggregation level

Parameters **aggregation_level** (*str*) – aggregation_level : str if further processing is to be done in zensus cell level ‘other’ else ‘dsitric’

Returns

- **CTS_per_district** (*pandas.DataFrame*) –
if aggregation = ‘district’ Profiles scaled up to annual demand
else 0
- **CTS_per_grid** (*pandas.DataFrame*) –
if aggregation = ‘district’ Profiles scaled up to annual demandd
else 0
- **CTS_per_zensus** (*pandas.DataFrame*) –
if aggregation = ‘district’ 0
else Profiles scaled up to annual demand

cts_demand_per_aggregation_level (*aggregation_level, scenario*)

Description: Create dataframe assigning the CTS demand curve to individual zensus cell based on their respective NUTS3 CTS curve

Parameters **aggregation_level** (*str*) – if further processing is to be done in zensus cell level ‘other’ else ‘dsitric’

Returns

- **CTS_per_district** (*pandas.DataFrame*) –
if aggregation = ‘district’ NUTS3 CTS profiles assigned to individual zensu cells and aggregated per district heat area id
else empty dataframe
- **CTS_per_grid** (*pandas.DataFrame*) –
if aggregation = ‘district’ NUTS3 CTS profiles assigned to individual zensu cells and aggregated per mv grid subst id
else empty dataframe
- **CTS_per_zensus** (*pandas.DataFrame*) –
if aggregation = ‘district’ empty dataframe

else NUTS3 CTS profiles assigned to individual zensu population id

```
class EgonEtragoHeatCts(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    bus_id

    p_set

    scn_name

class EgonEtragoTimeseriesIndividualHeating(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    bus_id

    dist_aggregated_mw

    scenario

class EgonIndividualHeatingPeakLoads(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    building_id

    scenario

    w_th

class EgonTimeseriesDistrictHeating(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    area_id

    dist_aggregated_mw

    scenario

class HeatTimeSeries(dependencies)
    Bases: egon.data.datasets.Dataset
```

Chooses heat demand profiles for each residential and CTS building

This dataset creates heat demand profiles in an hourly resolution. Time series for CTS buildings are created using the SLP-gas method implemented in the demandregio disaggregator with the function `export_etrago_cts_heat_profiles()` and stored in the database. Time series for residential buildings are created based on a variety of synthetical created individual demand profiles that are part of *DataBundle*. This method is described within the functions and in this publication:

C. Büttner, J. Amme, J. Endres, A. Malla, B. Schachler, I. Cußmann, Open modeling of electricity and heat demand curves for all residential buildings in Germany, Energy Informatics 5 (1) (2022) 21. doi:10.1186/s42162-022-00201-y.

Dependencies

- *DataBundle*
- *DemandRegio*
- *HeatDemandImport*
- *DistrictHeatingAreas*
- *Vg250*
- *ZensusMvGridDistricts*

- `hh_demand_buildings_setup`
- `WeatherData`

Resulting tables

- `demand.egon_timeseries_district_heating` is created and filled
- `demand.egon_etrage_heat_cts` is created and filled
- `demand.egon_heat_timeseries_selected_profiles` is created and filled
- `demand.egon_daily_heat_demand_per_climate_zone` is created and filled
- `boundaries.egon_map_zensus_climate_zones` is created and filled

```
name = 'HeatTimeSeries'
```

```
version = '0.0.7'
```

```
calculate_peak_load(df, scenario)
```

```
create_district_heating_profile(scenario, area_id)
```

Create heat demand profile for district heating grid including demands of households and service sector.

Parameters

- **scenario** (*str*) – Name of the selected scenario.
- **area_id** (*int*) – Index of the selected district heating grid

Returns `df` (*pandas.DataFrame*) – Hourly heat demand timeseries in MW for the selected district heating grid

```
create_district_heating_profile_python_like(scenario='eGon2035')
```

Creates profiles for all district heating grids in one scenario. Similar to `create_district_heating_profile` but faster and needs more RAM. The results are directly written into the database.

Parameters **scenario** (*str*) – Name of the selected scenario.

Returns *None*.

```
create_individual_heat_per_mv_grid(scenario='eGon2035', mv_grid_id=1564)
```

```
create_individual_heating_peak_loads(scenario='eGon2035')
```

```
create_individual_heating_profile_python_like(scenario='eGon2035')
```

```
create_timeseries_for_building(building_id, scenario)
```

Generates final heat demand timeseries for a specific building

Parameters

- **building_id** (*int*) – Index of the selected building
- **scenario** (*str*) – Name of the selected scenario.

Returns *pandas.DataFrame* – Hourly heat demand timeseries in MW for the selected building

```
district_heating(method='python')
```

```
export_etrage_cts_heat_profiles()
```

Export heat cts load profiles at mv substation level to `etrage`-table in the database

Returns *None*.

```
individual_heating_per_mv_grid(method='python')
```

```
individual_heating_per_mv_grid_100(method='python')
```

```
individual_heating_per_mv_grid_2035 (method='python')
individual_heating_per_mv_grid_tables (method='python')
store_national_profiles()
```

10.5.39 heat_etrago

hts_etrago

The central module creating heat demand time series for the eTraGo tool

```
class HtsEtragoTable (dependencies)
```

Bases: `egon.data.datasets.Dataset`

Collect heat demand time series for the eTraGo tool

This dataset collects data for individual and district heating demands and writes that into the tables that can be read by the eTraGo tool.

Dependencies

- `DistrictHeatingAreas`
- `HeatEtrago`
- `define_mv_grid_districts`
- `HeatPumps2035`
- `HeatTimeSeries`

Resulting tables

- `grid.egon_etrago_load` is extended
- `grid.egon_etrago_load_timeseries` is extended

```
name = 'HtsEtragoTable'
```

```
version = '0.0.6'
```

```
hts_to_etrago()
```

power_to_heat

The central module containing all code dealing with power to heat

```
assign_electrical_bus (heat_pumps, carrier, multiple_per_mv_grid=False)
```

Calculates heat pumps per electrical bus

Parameters

- **heat_pumps** (`pandas.DataFrame`) – Heat pumps including voltage level
- **multiple_per_mv_grid** (*boolean, optional*) – Choose if a district heating area can be supplied by multiple hvmv substations/mv grids. The default is False.

Returns `gdf` (`pandas.DataFrame`) – Heat pumps per electrical bus

```
assign_voltage_level (heat_pumps, carrier='heat_pump')
```

Assign voltage level to heat pumps

Parameters **heat_pumps** (`pandas.DataFrame`) – Heat pumps without voltage level

Returns `heat_pumps` (*pandas.DataFrame*) – Heat pumps including voltage level

insert_central_power_to_heat (*scenario='eGon2035'*)

Insert power to heat in district heating areas into database

Parameters `scenario` (*str, optional*) – Name of the scenario The default is 'eGon2035'.

Returns *None*.

insert_individual_power_to_heat (*scenario='eGon2035'*)

Insert power to heat into database

Parameters `scenario` (*str, optional*) – Name of the scenario The default is 'eGon2035'.

Returns *None*.

insert_power_to_heat_per_level (*heat_pumps, multiple_per_mv_grid, car-
rier='central_heat_pump', scenario='eGon2035'*)

Insert power to heat plants per grid level

Parameters

- `heat_pumps` (*pandas.DataFrame*) – Heat pumps in selected grid level
- `multiple_per_mv_grid` (*boolean*) – Choose if one district heating areas is supplied by one hvmv substation
- `scenario` (*str, optional*) – Name of the scenario The default is 'eGon2035'.

Returns *None*.

The central module containing all code dealing with heat sector in eTrago

class HeatEtrago (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Collect data related to the heat sector for the eTraGo tool

This dataset collects data from the heat sector and puts it into a format that is needed for the transmission grid optimisation within the tool eTraGo. It includes the creation of individual and central heat nodes, aggregates the heat supply technologies (apart from CHP) per medium voltage grid district and adds extendable heat stores to each bus. This data is then writing into the corresponding tables that are read by eTraGo.

Dependencies

- *HeatSupply*
- *define_mv_grid_districts*
- *EtragoSetup*
- *RenewableFeedin*
- *HeatTimeSeries*

Resulting tables

- *grid.egon_etrango_bus* is extended
- *grid.egon_etrango_link* is extended
- *grid.egon_etrango_link_timeseries* is extended
- *grid.egon_etrango_store* is extended
- *grid.egon_etrango_generator* is extended

name = 'HeatEtrago'


```
version = '0.0.10'
```

buses ()

Insert individual and district heat buses into eTraGo-tables

Returns *None*.

insert_buses (*carrier, scenario*)

Insert heat buses to etrago table

Heat buses are divided into central and individual heating

Parameters

- **carrier** (*str*) – Name of the carrier, either 'central_heat' or 'rural_heat'
- **scenario** (*str, optional*) – Name of the scenario.

insert_central_direct_heat (*scenario='eGon2035'*)

Insert renewable heating technologies (solar and geo thermal)

Parameters **scenario** (*str, optional*) – Name of the scenario The default is 'eGon2035'.

Returns *None*.

insert_central_gas_boilers (*scenario='eGon2035'*)

Inserts gas boilers for district heating to eTraGo-table

Parameters **scenario** (*str, optional*) – Name of the scenario. The default is 'eGon2035'.

Returns *None*.

insert_rural_gas_boilers (*scenario='eGon2035'*)

Inserts gas boilers for individual heating to eTraGo-table

Parameters **scenario** (*str, optional*) – Name of the scenario. The default is 'eGon2035'.

Returns *None*.

insert_store (*scenario, carrier*)

store ()

supply ()

Insert individual and district heat supply into eTraGo-tables

Returns *None*.

10.5.40 heat_supply

district_heating

The central module containing all code dealing with heat supply for district heating areas.

backup_gas_boilers (*scenario*)

Adds backup gas boilers to district heating grids.

Parameters **scenario** (*str*) – Name of the scenario.

Returns *Geopandas.GeoDataFrame* – List of gas boilers for district heating

backup_resistive_heaters (*scenario*)

Adds backup resistive heaters to district heating grids to meet target values of installed capacities.

Parameters **scenario** (*str*) – Name of the scenario.

Returns *Geopandas.GeoDataFrame* – List of gas boilers for district heating

capacity_per_district_heating_category (*district_heating_areas, scenario*)

Calculates target values per district heating category and technology

Parameters

- **district_heating_areas** (*geopandas.geodataframe.GeoDataFrame*) – District heating areas per scenario
- **scenario** (*str*) – Name of the scenario

Returns **capacity_per_category** (*pandas.DataFrame*) – Installed capacities per technology and size category

cascade_heat_supply (*scenario, plotting=True*)

Assigns supply strategy for district heating areas.

Different technologies are selected for three categories of district heating areas (small, medium and large annual demand). The technologies are prioritized according to Flexibilisierung der Kraft-Wärme-Kopplung; 2017; Forschungsstelle für Energiewirtschaft e.V. (FfE)

Parameters

- **scenario** (*str*) – Name of scenario
- **plotting** (*bool, optional*) – Choose if district heating supply is plotted. The default is True.

Returns **resulting_capacities** (*pandas.DataFrame*) – List of plants per district heating grid

cascade_per_technology (*areas, technologies, capacity_per_category, size_dh, max_geothermal_costs=2*)

Add plants of one technology supplying district heating

Parameters

- **areas** (*geopandas.geodataframe.GeoDataFrame*) – District heating areas which need to be supplied
- **technologies** (*pandas.DataFrame*) – List of supply technologies and their parameters
- **capacity_per_category** (*pandas.DataFrame*) – Target installed capacities per size-category
- **size_dh** (*str*) – Category of the district heating areas
- **max_geothermal_costs** (*float, optional*) – Maximal costs of MW geothermal in EUR/MW. The default is 2.

Returns

- **areas** (*geopandas.geodataframe.GeoDataFrame*) – District heating areas which need additional supply technologies
- **technologies** (*pandas.DataFrame*) – List of supply technologies and their parameters
- **append_df** (*pandas.DataFrame*) – List of plants per district heating grid for the selected technology

plot_heat_supply (*resulting_capacities*)

select_district_heating_areas (*scenario*)

Selects district heating areas per scenario and assigns size-category

Parameters **scenario** (*str*) – Name of the scenario

Returns **district_heating_areas** (*geopandas.geodataframe.GeoDataFrame*) – District heating areas per scenario

set_technology_data()

Set data per technology according to Kurzstudie KWK

Returns *pandas.DataFrame* – List of parameters per technology

geothermal

The module containing all code dealing with geothermal potentials and costs

Main source: Ableitung eines Korridors für den Ausbau der erneuerbaren Wärme im Gebäudebereich, Beuth Hochschule für Technik Berlin ifeu – Institut für Energie- und Umweltforschung Heidelberg GmbH Februar 2017

calc_geothermal_costs (*max_costs=inf, min_costs=0*)

calc_geothermal_potentials ()

calc_usable_geothermal_potential (*max_costs=2, min_costs=0*)

Calculate geothermal potentials close to district heating demands

Parameters

- **max_costs** (*float, optional*) – Maximum accepted costs for geo thermal in EUR/MW_{th}. The default is 2.
- **min_costs** (*float, optional*) – Minimum accepted costs for geo thermal in EUR/MW_{th}. The default is 0.

Returns *float* – Geothermal potential close to district heating areas in MW

potential_germany()

Calculates geothermal potentials for different investment costs.

The investment costs for geothermal district heating highly depend on the location because of different mass flows and drilling depths. This function calculates the geothermal potentials close to Germany for five different cost ranges. This data can be used in pypsa-eur-sec to optimise the share of geothermal district heating by considering different investment costs.

Returns *None*.

individual_heating

The central module containing all code dealing with individual heat supply.

The following main things are done in this module:

-
- Desaggregation of heat pump capacities to individual buildings
- Determination of minimum required heat pump capacity for pypsa-eur-sec

The determination of the minimum required heat pump capacity for pypsa-eur-sec takes place in the dataset 'Heat-PumpsPypsaEurSec'. The goal is to ensure that the heat pump capacities determined in pypsa-eur-sec are large enough to serve the heat demand of individual buildings after the disaggregation from a few nodes in pypsa-eur-sec to the individual buildings. To determine minimum required heat pump capacity per building the buildings heat peak load in the eGon100RE scenario is used (as pypsa-eur-sec serves as the scenario generator for the eGon100RE scenario; see [determine_minimum_hp_capacity_per_building\(\)](#) for information on how minimum required heat pump capacity is determined). As the heat peak load is not previously determined, it is as well done in the course of this task. Further, as determining heat peak load requires heat load profiles of the buildings to be set up, this task is

also utilised to set up heat load profiles of all buildings with heat pumps within a grid in the eGon100RE scenario used in eTraGo. The resulting data is stored in separate tables respectively a csv file:

- **input-pypsa-eur-sec/minimum_hp_capacity_mv_grid_100RE.csv:** This csv file contains minimum required heat pump capacity per MV grid in MW as input for pypsa-eur-sec. It is created within `export_min_cap_to_csv()`.
- **demand.egon_etrigo_timeseries_individual_heating:** This table contains aggregated heat load profiles of all buildings with heat pumps within an MV grid in the eGon100RE scenario used in eTraGo. It is created within `individual_heating_per_mv_grid_tables()`.
- **demand.egon_building_heat_peak_loads:** Mapping of peak heat demand and buildings including cell_id, building, area and peak load. This table is created in `delete_heat_peak_loads_100RE()`.

The disaggregation of heat pump capacities to individual buildings takes place in two separate datasets: ‘HeatPumps2035’ for eGon2035 scenario and ‘HeatPumps2050’ for eGon100RE. It is done separately because for one reason in case of the eGon100RE scenario the minimum required heat pump capacity per building can directly be determined using the heat peak load per building determined in the dataset ‘HeatPumpsPypsaEurSec’, whereas heat peak load data does not yet exist for the eGon2035 scenario. Another reason is, that in case of the eGon100RE scenario all buildings with individual heating have a heat pump whereas in the eGon2035 scenario buildings are randomly selected until the installed heat pump capacity per MV grid is met. All other buildings with individual heating but no heat pump are assigned a gas boiler.

In the ‘HeatPumps2035’ dataset the following things are done. First, the building’s heat peak load in the eGon2035 scenario is determined for sizing the heat pumps. To this end, heat load profiles per building are set up. Using the heat peak load per building the minimum required heat pump capacity per building is determined (see `determine_minimum_hp_capacity_per_building()`). Afterwards, the total heat pump capacity per MV grid is disaggregated to individual buildings in the MV grid, wherefore buildings are randomly chosen until the MV grid’s total heat pump capacity is reached (see `determine_buildings_with_hp_in_mv_grid()`). Buildings with PV rooftop plants are more likely to be assigned a heat pump. In case the minimum heat pump capacity of all chosen buildings is smaller than the total heat pump capacity of the MV grid but adding another building would exceed the total heat pump capacity of the MV grid, the remaining capacity is distributed to all buildings with heat pumps proportionally to the size of their respective minimum heat pump capacity. Therefore, the heat pump capacity of a building can be larger than the minimum required heat pump capacity. The generated heat load profiles per building are in a last step utilised to set up heat load profiles of all buildings with heat pumps within a grid as well as for all buildings with a gas boiler (i.e. all buildings with decentral heating system minus buildings with heat pump) needed in eTraGo. The resulting data is stored in the following tables:

- **demand.egon_hp_capacity_buildings:** This table contains the heat pump capacity of all buildings with a heat pump. It is created within `delete_hp_capacity_2035()`.
- **demand.egon_etrigo_timeseries_individual_heating:** This table contains aggregated heat load profiles of all buildings with heat pumps within an MV grid as well as of all buildings with gas boilers within an MV grid in the eGon100RE scenario used in eTraGo. It is created within `individual_heating_per_mv_grid_tables()`.
- **demand.egon_building_heat_peak_loads:** Mapping of heat demand time series and buildings including cell_id, building, area and peak load. This table is created in `delete_heat_peak_loads_2035()`.

In the ‘HeatPumps2050’ dataset the total heat pump capacity in each MV grid can be directly disaggregated to individual buildings, as the building’s heat peak load was already determined in the ‘HeatPumpsPypsaEurSec’ dataset. Also in contrast to the ‘HeatPumps2035’ dataset, all buildings with decentral heating system are assigned a heat pump, wherefore no random sampling of buildings needs to be conducted. The resulting data is stored in the following table:

- **demand.egon_hp_capacity_buildings:** This table contains the heat pump capacity of all buildings with a heat pump. It is created within `delete_hp_capacity_2035()`.

The following datasets from the database are mainly used for creation:

- *boundaries.egon_map_zensus_grid_districts:*

- *boundaries.egon_map_zensus_district_heating_areas*:
- ***demand.egon_peta_heat***: Table of annual heat load demand for residential and cts at census cell level from peta5.
- *demand.egon_heat_timeseries_selected_profiles*:
- *demand.egon_heat_idp_pool*:
- *demand.egon_daily_heat_demand_per_climate_zone*:
- ***boundaries.egon_map_zensus_mvgd_buildings***: A final mapping table including all buildings used for residential and cts, heat and electricity timeseries. Including census cells, mvgd bus_id, building type (osm or synthetic)
- *supply.egon_individual_heating*:
- ***demand.egon_cts_heat_demand_building_share***: Table including the mv substation heat profile share of all selected cts buildings for scenario eGon2035 and eGon100RE. This table is created within `cts_heat()`

What is the goal?

The goal is threefold. Primarily, heat pump capacity of individual buildings is determined as it is necessary for distribution grid analysis. Secondly, as heat demand profiles need to be set up during the process, the heat demand profiles of all buildings with individual heat pumps respectively gas boilers per MV grid are set up to be used in eTraGo. Thirdly, minimum heat pump capacity is determined as input for pypsa-eur-sec to avoid that heat pump capacity per building is too little to meet the heat demand after disaggregation to individual buildings.

What is the challenge?

The main challenge lies in the set up of heat demand profiles per building in `aggregate_residential_and_cts_profiles()` as it takes a lot of time and in grids with a high number of buildings requires a lot of RAM. Both runtime and RAM usage needed to be improved several times. To speed up the process, tasks are set up to run in parallel. This currently leads to a lot of connections being opened and at a certain point to a runtime error due to too many open connections.

What are central assumptions during the data processing?

Central assumption for determining minimum heat pump capacity and disaggregating heat pump capacity to individual buildings is that the required heat pump capacity is determined using an approach from the [network development plan](#) (pp.46-47) (see `determine_minimum_hp_capacity_per_building()`). There, the heat pump capacity is determined by multiplying the heat peak demand of the building by a minimum assumed COP of 1.7 and a flexibility factor of 24/18, taking into account that power supply of heat pumps can be interrupted for up to six hours by the local distribution grid operator. Another central assumption is, that buildings with PV rooftop plants are more likely to have a heat pump than other buildings (see `determine_buildings_with_hp_in_mv_grid()` for details)

Drawbacks and limitations of the data

In the eGon2035 scenario buildings with heat pumps are selected randomly with a higher probability for a heat pump for buildings with PV rooftop (see `determine_buildings_with_hp_in_mv_grid()` for details). Another limitation may be the sizing of the heat pumps, as in the eGon2035 scenario their size rigidly depends on the heat peak load and a fixed flexibility factor. During the coldest days of the year, heat pump flexibility strongly depends on this assumption and cannot be dynamically enlarged to provide more flexibility (or only slightly through larger heat storage units).

Notes

This module docstring is rather a dataset documentation. Once, a decision is made in ... the content of this module docstring needs to be moved to docs attribute of the respective dataset class.

```
class BuildingHeatPeakLoads (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    building_id
    peak_load_in_w
    scenario
    sector

class EgonEtragoTimeseriesIndividualHeating (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    bus_id
    carrier
    dist_aggregated_mw
    scenario

class EgonHpCapacityBuildings (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    building_id
    hp_capacity
    scenario

class HeatPumps2035 (dependencies)
    Bases: egon.data.datasets.Dataset

class HeatPumps2050 (dependencies)
    Bases: egon.data.datasets.Dataset

class HeatPumpsPypsaEurSec (dependencies)
    Bases: egon.data.datasets.Dataset

adapt_numpy_float64 (numpy_float64)
adapt_numpy_int64 (numpy_int64)

aggregate_residential_and_cts_profiles (mvgd, scenario)
    Gets residential and CTS heat demand profiles per building and aggregates them.
```

Parameters

- **mvgd** (*int*) – MV grid ID.
- **scenario** (*str*) – Possible options are eGon2035 or eGon100RE.

Returns *pd.DataFrame* – Table of demand profile per building. Column names are building IDs and index is hour of the year as int (0-8759).

calc_residential_heat_profiles_per_mvgd (mvgd, scenario)

Gets residential heat profiles per building in MV grid for either eGon2035 or eGon100RE scenario.

Parameters

- **mvgd** (*int*) – MV grid ID.
- **scenario** (*str*) – Possible options are eGon2035 or eGon100RE.

Returns

pd.DataFrame –

Heat demand profiles of buildings. Columns are:

- **zensus_population_id** [int] Zensus cell ID building is in.
- **building_id** [int] ID of building.
- **day_of_year** [int] Day of the year (1 - 365).
- **hour** [int] Hour of the day (1 - 24).
- **demand_ts** [float] Building's residential heat demand in MW, for specified hour of the year (specified through columns *day_of_year* and *hour*).

cascade_heat_supply_indiv (*scenario, distribution_level, plotting=True*)

Assigns supply strategy for individual heating in four steps.

- 1.) all small scale CHP are connected.
- 2.) If the supply can not meet the heat demand, solar thermal collectors are attached. This is not implemented yet, since individual solar thermal plants are not considered in eGon2035 scenario.
- 3.) If this is not suitable, the mv grid is also supplied by heat pumps.
- 4.) The last option are individual gas boilers.

Parameters

- **scenario** (*str*) – Name of scenario
- **plotting** (*bool, optional*) – Choose if individual heating supply is plotted. The default is True.

Returns **resulting_capacities** (*pandas.DataFrame*) – List of plants per mv grid

cascade_per_technology (*heat_per_mv, technologies, scenario, distribution_level, max_size_individual_chp=0.05*)

Add plants for individual heat. Currently only on mv grid district level.

Parameters

- **mv_grid_districts** (*geopandas.geodataframe.GeoDataFrame*) – MV grid districts including the heat demand
- **technologies** (*pandas.DataFrame*) – List of supply technologies and their parameters
- **scenario** (*str*) – Name of the scenario
- **max_size_individual_chp** (*float*) – Maximum capacity of an individual chp in MW

Returns

- **mv_grid_districts** (*geopandas.geodataframe.GeoDataFrame*) – MV grid district which need additional individual heat supply
- **technologies** (*pandas.DataFrame*) – List of supply technologies and their parameters
- **append_df** (*pandas.DataFrame*) – List of plants per mv grid for the selected technology

catch_missing_buidings (*buildings_decentral_heating, peak_load*)

Check for missing buildings and reduce the list of buildings with decentral heating if no peak loads available. This should only happen in case of cutout SH

Parameters

- **buildings_decentral_heating** (*list(int)*) – Array or list of buildings with decentral heating
- **peak_load** (*pd.Series*) – Peak loads of all building within the mvgrid

delete_heat_peak_loads_100RE()

Remove all heat peak loads for eGon100RE.

delete_heat_peak_loads_2035()

Remove all heat peak loads for eGon2035.

delete_hp_capacity(scenario)

Remove all hp capacities for the selected scenario

Parameters *scenario* (*string*) – Either eGon2035 or eGon100RE

delete_hp_capacity_100RE()

Remove all hp capacities for the selected eGon100RE

delete_hp_capacity_2035()

Remove all hp capacities for the selected eGon2035

delete_mvgd_ts(scenario)

Remove all mvgd ts for the selected scenario

Parameters *scenario* (*string*) – Either eGon2035 or eGon100RE

delete_mvgd_ts_100RE()

Remove all mvgd ts for the selected eGon100RE

delete_mvgd_ts_2035()

Remove all mvgd ts for the selected eGon2035

delete_pypsa_eur_sec_csv_file()

Delete pypsa eur sec minimum heat pump capacity csv before new run

desaggregate_hp_capacity(min_hp_cap_per_building, hp_cap_mv_grid)

Desaggregates the required total heat pump capacity to buildings.

All buildings are previously assigned a minimum required heat pump capacity. If the total heat pump capacity exceeds this, larger heat pumps are assigned.

Parameters

- **min_hp_cap_per_building** (*pd.Series*) –
Pandas series with minimum required heat pump capacity per building in MW.
- **hp_cap_mv_grid** (*float*) – Total heat pump capacity in MW in given MV grid.

Returns *pd.Series* – Pandas series with heat pump capacity per building in MW.

determine_buildings_with_hp_in_mv_grid(hp_cap_mv_grid, min_hp_cap_per_building)

Distributes given total heat pump capacity to buildings based on their peak heat demand.

Parameters

- **hp_cap_mv_grid** (*float*) – Total heat pump capacity in MW in given MV grid.
- **min_hp_cap_per_building** (*pd.Series*) –
Pandas series with minimum required heat pump capacity per building in MW.

Returns *pd.Index(int)* – Building IDs (as int) of buildings to get heat demand time series for.

determine_hp_cap_buildings_eGon100RE()

Main function to determine HP capacity per building in eGon100RE scenario.

determine_hp_cap_buildings_eGon100RE_per_mvgd(mv_grid_id)

Determines HP capacity per building in eGon100RE scenario.

In eGon100RE scenario all buildings without district heating get a heat pump.

Returns *pd.Series* – Pandas series with heat pump capacity per building in MW.

determine_hp_cap_buildings_eGon2035_per_mvgd (*mv_grid_id*, *peak_heat_demand*, *building_ids*)

Determines which buildings in the MV grid will have a HP (buildings with PV rooftop are more likely to be assigned) in the eGon2035 scenario, as well as their respective HP capacity in MW.

Parameters

- **mv_grid_id** (*int*) – ID of MV grid.
- **peak_heat_demand** (*pd.Series*) – Series with peak heat demand per building in MW. Index contains the building ID.
- **building_ids** (*pd.Index(int)*) – Building IDs (as int) of buildings with decentral heating system in given MV grid.

determine_hp_cap_peak_load_mvgd_ts_2035 (*mvgd_ids*)

Main function to determine HP capacity per building in eGon2035 scenario. Further, creates heat demand time series for all buildings with heat pumps in MV grid, as well as for all buildings with gas boilers, used in eTraGo.

Parameters *mvgd_ids* (*list(int)*) – List of MV grid IDs to determine data for.

determine_hp_cap_peak_load_mvgd_ts_pypsa_eur_sec (*mvgd_ids*)

Main function to determine minimum required HP capacity in MV for pypsa-eur-sec. Further, creates heat demand time series for all buildings with heat pumps in MV grid in eGon100RE scenario, used in eTraGo.

Parameters *mvgd_ids* (*list(int)*) – List of MV grid IDs to determine data for.

determine_min_hp_cap_buildings_pypsa_eur_sec (*peak_heat_demand*, *building_ids*)

Determines minimum required HP capacity in MV grid in MW as input for pypsa-eur-sec.

Parameters

- **peak_heat_demand** (*pd.Series*) – Series with peak heat demand per building in MW. Index contains the building ID.
- **building_ids** (*pd.Index(int)*) – Building IDs (as int) of buildings with decentral heating system in given MV grid.

Returns *float* – Minimum required HP capacity in MV grid in MW.

determine_minimum_hp_capacity_per_building (*peak_heat_demand*, *flexibility_factor=1.3333333333333333*, *cop=1.7*)

Determines minimum required heat pump capacity.

Parameters

- **peak_heat_demand** (*pd.Series*) – Series with peak heat demand per building in MW. Index contains the building ID.
- **flexibility_factor** (*float*) – Factor to overdimension the heat pump to allow for some flexible dispatch in times of high heat demand. Per default, a factor of 24/18 is used, to take into account

Returns *pd.Series* – Pandas series with minimum required heat pump capacity per building in MW.

export_min_cap_to_csv (*df_hp_min_cap_mv_grid_pypsa_eur_sec*)

Export minimum capacity of heat pumps for pypsa eur sec to csv

export_to_db (*df_peak_loads_db*, *df_heat_mvgd_ts_db*, *drop=False*)

Function to export the collected results of all MVGDs per bulk to DB.

Parameters

df_peak_loads_db [pd.DataFrame] Table of building peak loads of all MVGDs per bulk

df_heat_mvgd_ts_db [pd.DataFrame] Table of all aggregated MVGD profiles per bulk

drop [boolean] Drop and recreate table if True

get_buildings_with_decentral_heat_demand_in_mv_grid (*mvgd, scenario*)

Returns building IDs of buildings with decentral heat demand in given MV grid.

As cells with district heating differ between scenarios, this is also depending on the scenario. CTS and residential have to be retrieved separately as some residential buildings only have electricity but no heat demand. This does not occur in CTS.

Parameters

- **mvgd** (*int*) – ID of MV grid.
- **scenario** (*str*) – Name of scenario. Can be either “eGon2035” or “eGon100RE”.

Returns *pd.Index(int)* – Building IDs (as int) of buildings with decentral heating system in given MV grid. Type is pandas Index to avoid errors later on when it is used in a query.

get_cts_buildings_with_decentral_heat_demand_in_mv_grid (*scenario, mv_grid_id*)

Returns building IDs of buildings with decentral CTS heat demand in given MV grid.

As cells with district heating differ between scenarios, this is also depending on the scenario.

Parameters

- **scenario** (*str*) – Name of scenario. Can be either “eGon2035” or “eGon100RE”.
- **mv_grid_id** (*int*) – ID of MV grid.

Returns *pd.Index(int)* – Building IDs (as int) of buildings with decentral heating system in given MV grid. Type is pandas Index to avoid errors later on when it is used in a query.

get_daily_demand_share (*mvgd*)

per census cell :Parameters: **mvgd** (*int*) – MVGD id

Returns **df_daily_demand_share** (*pd.DataFrame*) – Daily annual demand share per census cell. Columns of the dataframe are *zensus_population_id*, *day_of_year* and *daily_demand_share*.

get_daily_profiles (*profile_ids*)

Parameters **profile_ids** (*list(int)*) – daily heat profile ID’s

Returns **df_profiles** (*pd.DataFrame*) – Residential daily heat profiles. Columns of the dataframe are *idp*, *house*, *temperature_class* and *hour*.

get_heat_peak_demand_per_building (*scenario, building_ids*)

get_peta_demand (*mvgd, scenario*)

Retrieve annual peta heat demand for residential buildings for either eGon2035 or eGon100RE scenario.

Parameters

- **mvgd** (*int*) – MV grid ID.
- **scenario** (*str*) – Possible options are eGon2035 or eGon100RE

Returns **df_peta_demand** (*pd.DataFrame*) – Annual residential heat demand per building and scenario. Columns of the dataframe are *zensus_population_id* and *demand*.

get_residential_buildings_with_decentral_heat_demand_in_mv_grid (*scenario, mv_grid_id*)

Returns building IDs of buildings with decentral residential heat demand in given MV grid.

As cells with district heating differ between scenarios, this is also depending on the scenario.

Parameters

- **scenario** (*str*) – Name of scenario. Can be either “eGon2035” or “eGon100RE”.
- **mv_grid_id** (*int*) – ID of MV grid.

Returns *pd.Index(int)* – Building IDs (as int) of buildings with decentral heating system in given MV grid. Type is pandas Index to avoid errors later on when it is used in a query.

get_residential_heat_profile_ids (*mvgd*)

Retrieve 365 daily heat profiles ids per residential building and selected mvgd.

Parameters **mvgd** (*int*) – ID of MVGD

Returns **df_profiles_ids** (*pd.DataFrame*) – Residential daily heat profile ID’s per building. Columns of the dataframe are *zensus_population_id*, *building_id*, *selected_idp_profiles*, *buildings* and *day_of_year*.

get_total_heat_pump_capacity_of_mv_grid (*scenario, mv_grid_id*)

Returns total heat pump capacity per grid that was previously defined (by NEP or pypsa-eur-sec).

Parameters

- **scenario** (*str*) – Name of scenario. Can be either “eGon2035” or “eGon100RE”.
- **mv_grid_id** (*int*) – ID of MV grid.

Returns *float* – Total heat pump capacity in MW in given MV grid.

get_zensus_cells_with_decentral_heat_demand_in_mv_grid (*scenario, mv_grid_id*)

Returns zensus cell IDs with decentral heating systems in given MV grid.

As cells with district heating differ between scenarios, this is also depending on the scenario.

Parameters

- **scenario** (*str*) – Name of scenario. Can be either “eGon2035” or “eGon100RE”.
- **mv_grid_id** (*int*) – ID of MV grid.

Returns *pd.Index(int)* – Zensus cell IDs (as int) of buildings with decentral heating systems in given MV grid. Type is pandas Index to avoid errors later on when it is used in a query.

plot_heat_supply (*resulting_capacities*)

split_mvghs_into_bulks (*n, max_n, func*)

Generic function to split task into multiple parallel tasks, dividing the number of MVGDs into even bulks.

Parameters

- **n** (*int*) – Number of bulk
- **max_n** (*int*) – Maximum number of bulks
- **func** (*function*) – The function which is then called with the list of MVGD as parameter.

The central module containing all code dealing with heat supply data

class EgonDistrictHeatingSupply (***kwargs*)

Bases: *sqlalchemy.ext.declarative.api.Base*

capacity

carrier

category

district_heating_id

geometry

index

scenario

```
class EgonIndividualHeatingSupply (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
```

capacity

carrier

category

geometry

index

mv_grid_id

scenario

```
class HeatSupply (dependencies)
    Bases: egon.data.datasets.Dataset
```

Select and store heat supply technologies for individual and district heating

This dataset distributes heat supply technologies to each district heating grid and individual supplies buildings per medium voltage grid district. National installed capacities are predefined from external sources within *ScenarioCapacities*. The further distribution is done using a cascade that follows a specific order of supply technologies and the heat demand.

Dependencies

- *DataBundle*
- *DistrictHeatingAreas*
- *ZensusMvGridDistricts*
- *Chp*

Resulting tables

- *demand.egon_district_heating* is created and filled
- *demand.egon_individual_heating* is created and filled

name = 'HeatSupply'

version = '0.0.8'

```
create_tables()
    Create tables for district heating areas
```

Returns *None*

```
district_heating()
    Insert supply for district heating areas
```

Returns *None.*

```
individual_heating()
    Insert supply for individual heating
```

Returns *None.*

10.5.41 hydrogen_etrago

bus

The central module containing all code dealing with the hydrogen buses

In this module, the functions allowing to create the H2 buses in Germany for eTraGo are to be found. The H2 buses in the neighbouring countries (only present in eGon100RE) are defined in *pypsaetursec*. In both scenarios, there are two types of H2 buses in Germany:

- H2_grid buses: defined in *insert_H2_buses_from_CH4_grid()*; these buses are located at the places of the CH4 buses.
- H2_saltcavern buses: defined in *insert_H2_buses_from_saltcavern()*; these buses are located at the intersection of AC buses and potential H2 saltcaverns.

insert_H2_buses_from_CH4_grid (*gdf, carrier, target, scn_name*)

Insert the H2 buses based on CH4 grid into the database.

At each CH4 location, respectively at each intersection of the CH4 grid, a H2 bus is created.

Parameters

- **gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing the empty bus data.
- **carrier** (*str*) – Name of the carrier.
- **target** (*dict*) – Target schema and table information.
- **scn_name** (*str*) – Name of the scenario.

Returns *None*

insert_H2_buses_from_saltcavern (*gdf, carrier, sources, target, scn_name*)

Insert the H2 buses based on saltcavern locations into the database.

These buses are located at the intersection of AC buses and potential H2 saltcaverns.

Parameters

- **gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing the empty bus data.
- **carrier** (*str*) – Name of the carrier.
- **sources** (*dict*) – Sources schema and table information.
- **target** (*dict*) – Target schema and table information.
- **scn_name** (*str*) – Name of the scenario.

Returns *None*

insert_hydrogen_buses (*scenario='eGon2035'*)

Insert hydrogen buses into the database (in etrigo table)

Hydrogen buses are inserted into the database using the functions:

- *insert_H2_buses_from_CH4_grid()* for H2_grid buses
- *insert_H2_buses_from_saltcavern()* for the H2_saltcavern buses

Parameters **scenario** (*str, optional*) – Name of the scenario, the default is 'eGon2035'.

Returns *None*

insert_hydrogen_buses_eGon100RE ()

Copy H2 buses from the eGon2035 to the eGon100RE scenario.

Returns *None*

h2_grid

The central module containing all code dealing with the H2 grid in eGon100RE

The H2 grid, present only in eGon100RE, is composed of two parts:

- a fixed part with the same topology as the CH4 grid and with carrier 'H2_retrofit' corresponding to the retrofitted share of the CH4 grid into a hydrogen grid,
- an extendable part with carrier 'H2_gridextension', linking each H2_salcavern bus to the closest H2_grid bus: this part has no capacity ($p_{nom} = 0$) but it can be extended.

As for the CH4 grid, the H2 pipelines are modelled by PyPSA links.

insert_h2_pipelines ()

Insert hydrogen grid (H2 links) into the database for eGon100RE.

Insert the H2 grid by executing the following steps:

- Copy the CH4 links in Germany from eGon2035
- **Overwrite the followings columns:**
 - bus0 and bus1 using the grid.egon_etrageo_ch4_h2 table
 - carrier, scn_name
 - p_nom: the value attributed there corresponds to the share of p_nom of the specific pipeline that could be retrofitted into H2 pipeline. This share is the same for every pipeline and is calculated in the PyPSA-eur-sec run.
- Create new extendable pipelines to link the existing grid to the H2_salcavern buses
- Clean database
- Attribute link_id to the links
- Insert into the database

Returns *None*

h2_to_ch4

Module containing the definition of the links between H2 and CH4 buses

In this module the functions used to define and insert the links between H2 and CH4 buses into the database are to be found. These links are modelling:

- Methanisation (carrier name: 'H2_to_CH4'): technology to produce CH4 from H2
- H2_feedin: Injection of H2 into the CH4 grid
- Steam Methane Reaction (SMR, carrier name: 'CH4_to_H2'): technology to produce CH4 from H2

H2_CH4_mix_energy_fractions (x, T=25, p=50)

Calculate the fraction of H2 with respect to energy in a H2 CH4 mixture.

Given the volumetric fraction of H2 in a H2 and CH4 mixture, the fraction of H2 with respect to energy is calculated with the ideal gas mixture law. Beware, that changing the fraction of H2 changes the overall energy

within a specific volume of the mixture. If H2 is fed into CH4, the pipeline capacity (based on energy) therefore decreases if the volumetric flow does not change. This effect is neglected in eGon. At 15 vol% H2 the decrease in capacity equals about 10 % if volumetric flow does not change.

Parameters

- **x** (*float*) – Volumetric fraction of H2 in the mixture
- **T** (*int, optional*) – Temperature of the mixture in °C, by default 25
- **p** (*int, optional*) – Pressure of the mixture in bar, by default 50

Returns *float* – Fraction of H2 in mixture with respect to energy (LHV)

insert_h2_to_ch4_eGon100RE()

Copy H2/CH4 links from the eGon2035 to the eGon100RE scenario.

insert_h2_to_ch4_to_h2()

Inserts methanisation, feedin and SMR links into the database

Define the potentials for methanisation and Steam Methane Reaction (SMR) modelled as extendable links as well as the H2 feedin capacities modelled as non extendable links and insert all of them into the database. These tree technologies are connecting CH4 and H2_grid buses only.

The capacity of the H2_feedin links is considered as constant and calculated as the sum of the capacities of the CH4 links connected to the CH4 bus multiplied by the H2 energy share allowed to be fed in. This share is calculated in the function `H2_CH4_mix_energy_fractions()`.

Returns *None*

power_to_h2

Module containing the definition of the AC grid to H2 links

In this module the functions used to define and insert the links between H2 and AC buses into the database are to be found. These links are modelling:

- Electrolysis (carrier name: 'power_to_H2'): technology to produce H2 from AC
- Fuel cells (carrier name: 'H2_to_power'): technology to produce power from H2

insert_power_to_h2_to_power(scn_name='eGon2035')

Insert electrolysis and fuel cells capacities into the database.

The potentials for power-to-H2 in electrolysis and H2-to-power in fuel cells are created between each H2 bus (H2_grid and H2_saltcavern) and its closest HV power bus. These links are extendable. For the electrolysis, if the distance between the AC and the H2 bus is > 500m, the maximum capacity of the installation is limited to 1 MW.

Parameters **scn_name** (*str*) – Name of the scenario

Returns *None*

insert_power_to_h2_to_power_eGon100RE()

Copy H2/power links from the eGon2035 to the eGon100RE scenario.

Returns *None*

map_buses(scn_name)

Map H2 buses to nearest HV AC bus.

Parameters **scn_name** (*str*) – Name of the scenario.

Returns **gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame with connected buses.

storage

The central module containing all code dealing with H2 stores in Germany

This module contains the functions used to insert the two types of H2 store potentials in Germany:

- H2 overground stores (carrier: 'H2_overground'): steel tanks at every H2_grid bus
- H2 underground stores (carrier: 'H2_underground'): saltcavern store at every H2_saltcavern bus. NB: the saltcavern locations define the H2_saltcavern buses locations.

All these stores are modelled as extendable PyPSA stores.

calculate_and_map_saltcavern_storage_potential()
Calculate site specific storage potential based on InSpEE-DS report.

Returns *None*

insert_H2_overground_storage(*scn_name='eGon2035'*)
Insert H2_overground stores into the database.

Insert extendable H2_overground stores (steel tanks) at each H2_grid bus.

Returns *None*

insert_H2_saltcavern_storage(*scn_name='eGon2035'*)
Insert H2_underground stores into the database.

Insert extendable H2_underground stores (saltcavern potentials) at every H2_saltcavern bus.

Returns *None*

insert_H2_storage_eGon100RE()
Copy H2 storage from the eGon2035 to the eGon100RE scenario.

Returns *None*

write_saltcavern_potential()
Write saltcavern potentials into the database

Returns *None*

The central module containing the definitions of the datasets linked to H2

This module contains the definitions of the datasets linked to the hydrogen sector in eTraGo in Germany.

In the eGon2035 scenario, there is no H2 bus abroad, so technologies linked to the hydrogen sector are present only in Germany.

In the eGon100RE scenario, the potential and installed capacities abroad arise from the PyPSA-eur-sec run. For this reason, this module focuses only on the hydrogen related components in Germany, and the module *pypsaeursec* on the hydrogen related components abroad.

class HydrogenBusEtrago(*dependencies*)
Bases: *egon.data.datasets.Dataset*

Insert the H2 buses into the database for Germany

Insert the H2 buses in Germany into the database for the scenarios eGon2035 and eGon100RE by executing successively the functions *calculate_and_map_saltcavern_storage_potential*, *insert_hydrogen_buses* and *insert_hydrogen_buses_eGon100RE*.

Dependencies

- *SaltcavernData*
- *GasNodesAndPipes*

- *SubstationVoronoi*

Resulting

- *grid.egon_etrango_bus* is extended

name = 'HydrogenBusEtrango'

version = '0.0.1'

class HydrogenGridEtrango (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Insert the H2 grid in Germany into the database for eGon100RE

Insert the H2 links (pipelines) into Germany in the database for the scenario eGon100RE by executing the function *insert_h2_pipelines*.

Dependencies

- *SaltcavernData*
- *GasNodesAndPipes*
- *SubstationVoronoi*
- *GasAreaseGon2035*
- *PypsaEurSec*
- *HydrogenBusEtrango*

Resulting

- *grid.egon_etrango_link* is extended

name = 'HydrogenGridEtrango'

version = '0.0.2'

class HydrogenMethaneLinkEtrango (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Insert the methanisation, feed in and SMR into the database

Insert the the methanisation, feed in (only in eGon2035) and Steam Methane Reaction (SMR) links in Germany into the database for the scenarios eGon2035 and eGon100RE by executing successively the functions *insert_h2_to_ch4_to_h2* and *insert_h2_to_ch4_eGon100RE*.

Dependencies

- *SaltcavernData*
- *GasNodesAndPipes*
- *SubstationVoronoi*
- *HydrogenBusEtrango*
- *HydrogenGridEtrango*
- *HydrogenPowerLinkEtrango*

Resulting

- *grid.egon_etrango_link* is extended

name = 'HydrogenMethaneLinkEtrango'

version = '0.0.5'

class HydrogenPowerLinkEtrago (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Insert the electrolysis and the fuel cells into the database

Insert the the electrolysis and the fuel cell links in Germany into the database for the scenarios eGon2035 and eGon100RE by executing successively the functions *insert_power_to_h2_to_power* and *insert_power_to_h2_to_power_eGon100RE*.

Dependencies

- *SaltcavernData*
- *GasNodesAndPipes*
- *SubstationVoronoi*
- *HydrogenBusEtrago*
- *HydrogenGridEtrago*

Resulting

- *grid.egon_etrango_link* is extended

name = 'HydrogenPowerLinkEtrago'

version = '0.0.4'

class HydrogenStoreEtrago (*dependencies*)

Bases: *egon.data.datasets.Dataset*

Insert the H2 stores into the database for Germany

Insert the H2 stores in Germany into the database for the scenarios eGon2035 and eGon100RE:

- H2 overground stores or steel tanks at each H2_grid bus with the function *insert_H2_overground_storage* for the scenario eGon2035,
- H2 underground stores or saltcavern stores at each H2_saltcavern bus with the function *insert_H2_saltcavern_storage* for the scenario eGon2035,
- H2 stores (overground and underground) for the scenario eGon100RE with the function *insert_H2_storage_eGon100RE*.

Dependencies

- *SaltcavernData*
- *GasNodesAndPipes*
- *SubstationVoronoi*
- *HydrogenBusEtrago*
- *HydrogenGridEtrago*
- *GasNodesAndPipes*

Resulting

- *grid.egon_etrango_store* is extended

name = 'HydrogenStoreEtrago'

version = '0.0.3'

10.5.42 industrial_sites

The central module containing all code dealing with the spatial distribution of industrial electricity demands. Industrial demands from DemandRegio are distributed from nuts3 level down to osm landuse polygons and/or industrial sites also identified within this processing step bringing three different inputs together.

```
class HotmapsIndustrialSites (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    address
    city
    citycode
    companyname
    country
    datasource
    emissions_eprtr_2014
    emissions_ets_2014
    excess_heat_100_200C
    excess_heat_200_500C
    excess_heat_500C
    excess_heat_total
    fuel_demand
    geom
    location
    production
    siteid
    sitename
    subsector
    wz

class IndustrialSites (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    address
    companyname
    geom
    id
    nuts3
    subsector
    wz

class MergeIndustrialSites (dependencies)
    Bases: egon.data.datasets.Dataset
```

```
class SchmidtIndustrialSites(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    annual_tonnes
    application
    capacity_production
    geom
    id
    landkreis_number
    lat
    lon
    plant
    wz

class SeenergiesIndustrialSites(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    address
    companyname
    country
    electricitydemand_tj
    eu28
    excess_heat
    fueldemand_tj
    geom
    globalid
    lat
    level_1_pj
    level_1_r_pj
    level_1_r_tj
    level_1_tj
    level_2_pj
    level_2_r_pj
    level_2_r_tj
    level_2_tj
    level_3_pj
    level_3_r_pj
    level_3_r_tj
    level_3_tj
    lon
```

```

nuts1
nuts3
objectid
siteid
subsector
wz
create_tables ()
    Create tables for industrial sites and distributed industrial demands :returns: None.
download_hotmaps ()
    Download csv file on hotmap's industrial sites.
download_import_industrial_sites ()
    Wraps different functions to create tables, download csv files containing information on industrial sites in Germany and write this data to the local postgresql database

    Returns None.
download_seenergies ()
    Download csv file on s-eenergies' industrial sites.
hotmaps_to_postgres ()
    Import hotmaps data to postgres database
map_nuts3 ()
    Match resulting industrial sites with nuts3 codes and fill column 'nuts3'

    Returns None.
merge_inputs ()
    Merge and clean data from different sources (hotmaps, seenergies, Thesis Schmidt)
schmidt_to_postgres ()
    Import data from Thesis by Danielle Schmidt to postgres database
seenergies_to_postgres ()
    Import seenergies data to postgres database

```

10.5.43 industry

temporal

The central module containing all code dealing with processing timeseries data using demandregio

```

calc_load_curves_ind_osm (scenario)
    Temporal disaggregate electrical demand per osm industrial landuse area.

    Parameters scenario (str) – Scenario name.

    Returns pandas.DataFrame – Demand timeseries of industry allocated to osm landuse areas and aggregated per substation id
calc_load_curves_ind_sites (scenario)
    Temporal disaggregation of load curves per industrial site and industrial subsector.

    Parameters scenario (str) – Scenario name.

```

Returns *pandas.DataFrame* – Demand timeseries of industry allocated to industrial sites and aggregated per substation id and industrial subsector

identify_bus (*load_curves*, *demand_area*)

Identify the grid connection point for a consumer by determining its grid level based on the time series' peak load and the spatial intersection to my grid districts or ehv voronoi cells.

Parameters

- **load_curves** (*pandas.DataFrame*) – Demand timeseries per demand area (e.g. osm landuse area, industrial site)
- **demand_area** (*pandas.DataFrame*) – Dataframe with id and geometry of areas where an industrial demand is assigned to, such as osm landuse areas or industrial sites.

Returns *pandas.DataFrame* – Aggregated industrial demand timeseries per bus

identify_voltage_level (*df*)

Identify the voltage_level of a grid component based on its peak load and defined thresholds.

Parameters *df* (*pandas.DataFrame*) – Data frame containing information about peak loads

Returns *pandas.DataFrame* – Data frame with an additional column with voltage level

insert_osm_ind_load ()

Inserts electrical industry loads assigned to osm landuse areas to the database.

Returns *None*.

insert_sites_ind_load ()

Inserts electrical industry loads assigned to osm landuse areas to the database.

Returns *None*.

The central module containing all code dealing with the spatial distribution of industrial electricity demands. Industrial demands from DemandRegio are distributed from nuts3 level down to osm landuse polygons and/or industrial sites also identified within this processing step bringing three different inputs together.

class DemandCurvesOsmIndustry (***kwargs*)

Bases: *sqlalchemy.ext.declarative.api.Base*

bus

p_set

scn_name

class DemandCurvesOsmIndustryIndividual (***kwargs*)

Bases: *sqlalchemy.ext.declarative.api.Base*

bus_id

demand

osm_id

p_set

peak_load

scn_name

voltage_level

class DemandCurvesSitesIndustry (***kwargs*)

Bases: *sqlalchemy.ext.declarative.api.Base*

```

    bus
    p_set
    scn_name
    wz
class DemandCurvesSitesIndustryIndividual(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus_id
    demand
    p_set
    peak_load
    scn_name
    site_id
    voltage_level
    wz
class EgonDemandRegioOsmIndElectricity(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    demand
    id
    osm_id
    scenario
    wz
class EgonDemandRegioSitesIndElectricity(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    demand
    industrial_sites_id
    scenario
    wz
class IndustrialDemandCurves(dependencies)
    Bases: egon.data.datasets.Dataset
create_tables()
    Create tables for industrial sites and distributed industrial demands :returns: None.
industrial_demand_distr()
    Distribute electrical demands for industry to osm landuse polygons and/or industrial sites, identified earlier in the process. The demands per subsector on nuts3-level from demandregio are distributed linearly to the area of the corresponding landuse polygons or evenly to identified industrial sites.

    Returns None.

```

10.5.44 loadarea

OSM landuse extraction and load areas creation.

Landuse

- Landuse data is extracted from OpenStreetMap: residential, retail, industrial, Agricultural
- Data is cut with German borders (VG 250), data outside is dropped
- Invalid geometries are fixed
- Results are stored in table *openstreetmap.osm_landuse*

Load Areas

TBD

Note: industrial demand contains: * voltage levels 4-7 * only demand from ind. sites+osm located in LA!

```
class LoadArea (dependencies)
    Bases: egon.data.datasets.Dataset

class OsmLanduse (dependencies)
    Bases: egon.data.datasets.Dataset

class OsmPolygonUrban (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    area_ha
    geom
    id
    name
    osm_id
    sector
    sector_name
    tags
    vg250

census_cells_melt ()
    Melt all census cells: buffer, union, unbuffer

create_landuse_table ()
    Create tables for landuse data :returns: None.

drop_temp_tables ()

execute_sql_script (script)
    Execute SQL script

    Parameters script (str) – Filename of script

loadareas_add_demand_cts ()
    Adds consumption and peak load to load areas for CTS

loadareas_add_demand_hh ()
    Adds consumption and peak load to load areas for households

loadareas_add_demand_ind ()
    Adds consumption and peak load to load areas for industry
```


loadareas_create()

Create load areas from merged OSM landuse and census cells:

- Cut Loadarea with MV Griddistrict
- Identify and exclude Loadarea smaller than 100m².
- Generate Centre of Loadareas with Centroid and PointOnSurface.
- Calculate population from Census 2011.
- Cut all 4 OSM sectors with MV Griddistricts.
- Calculate statistics like NUTS and AGS code.
- Check for Loadareas without AGS code.

osm_landuse_census_cells_melt()

Melt OSM landuse areas and census cells

osm_landuse_melt()

Melt all OSM landuse areas by: buffer, union, unbuffer

10.5.45 low_flex_scenario

The central module to create low flex scenarios

class LowFlexScenario (*dependencies*)

Bases: *egon.data.datasets.Dataset*

10.5.46 osm

The central module containing all code dealing with importing OSM data.

This module either directly contains the code dealing with importing OSM data, or it re-exports everything needed to handle it. Please refrain from importing code from any modules below this one, because it might lead to unwanted behaviour.

If you have to import code from a module below this one because the code isn't exported from this module, please file a bug, so we can fix this.

class OpenStreetMap (*dependencies*)

Bases: *egon.data.datasets.Dataset*

add_metadata()

Writes metadata JSON string into table comment.

download()

Download OpenStreetMap *.pbf* file.

modify_tables()

Adjust primary keys, indices and schema of OSM tables.

- The Column "id" is added and used as the new primary key.
- Indices (GIST, GIN) are reset
- The tables are moved to the schema configured as the "output_schema".

to_postgres (*cache_size=4096*)

Import OSM data from a Geofabrik *.pbf* file into a PostgreSQL database.

Parameters *cache_size* (*int, optional*) – Memory used during data import

10.5.47 osm_buildings_streets

Filtered and preprocessed buildings, streets and amenities from OpenStreetMap (OSM)

This dataset on buildings and amenities is required by several tasks in the pipeline, such as the distribution of household demand profiles or PV home systems to buildings. This data is enriched by population and apartments from Zensus 2011. Those derived datasets and the data on streets will be used in the DIstribution Network Generator [:ref:'dingo'<https://github.com/openego/dingo>](https://github.com/openego/dingo) e.g. to cluster loads and create low voltage grids.

Details and Steps

- Extract buildings and filter using relevant tags, e.g. residential and commercial, see script *osm_buildings_filter.sql* for the full list of tags. Resulting tables: * All buildings: *openstreetmap.osm_buildings* * Filtered buildings: *openstreetmap.osm_buildings_filtered* * Residential buildings: *openstreetmap.osm_buildings_residential*
- Extract amenities and filter using relevant tags, e.g. shops and restaurants, see script *osm_amenities_shops_preprocessing.sql* for the full list of tags. Resulting table: *openstreetmap.osm_amenities_shops_filtered*
- Create a mapping table for building's osm IDs to the Zensus cells the building's centroid is located in. Resulting tables: * *boundaries.egon_map_zensus_buildings_filtered* (filtered) * *boundaries.egon_map_zensus_buildings_residential* (residential only)
- Enrich each building by number of apartments from Zensus table *society.egon_destatis_zensus_apartment_building_population_per_ha* by splitting up the cell's sum equally to the buildings. In some cases, a Zensus cell does not contain buildings but there's a building nearby which the no. of apartments is to be allocated to. To make sure apartments are allocated to at least one building, a radius of 77m is used to catch building geometries.
- Split filtered buildings into 3 datasets using the amenities' locations: temporary tables are created in script *osm_buildings_temp_tables.sql* the final tables in *osm_buildings_amenities_results.sql*. Resulting tables: * Buildings w/ amenities: *openstreetmap.osm_buildings_with_amenities* * Buildings w/o amenities: *openstreetmap.osm_buildings_without_amenities* * Amenities not allocated to buildings: *openstreetmap.osm_amenities_not_in_buildings*
- Extract streets (OSM ways) and filter using relevant tags, e.g. highway=secondary, see script *osm_ways_preprocessing.sql* for the full list of tags. Additionally, each way is split into its line segments and their lengths is retained. Resulting tables: * Filtered streets: *openstreetmap.osm_ways_preprocessed* * Filtered streets w/ segments: *openstreetmap.osm_ways_with_segments*

Notes

This module docstring is rather a dataset documentation. Once, a decision is made in ... the content of this module docstring needs to be moved to docs attribute of the respective dataset class.

```
class OsmBuildingsStreets (dependencies)
    Bases: egon.data.datasets.Dataset

    add_metadata ()

    create_buildings_filtered_all_zensus_mapping ()

    create_buildings_filtered_zensus_mapping ()

    create_buildings_residential_zensus_mapping ()

    create_buildings_temp_tables ()

    drop_temp_tables ()
```

execute_sql_script (*script*)

Execute SQL script

Parameters *script* (*str*) – Filename of script

extract_amenities ()

extract_buildings_filtered_amenities ()

extract_buildings_w_amenities ()

extract_buildings_wo_amenities ()

extract_ways ()

filter_buildings ()

filter_buildings_residential ()

preprocessing ()

10.5.48 osmtgmod

substation

The central module containing code to create substation tables

```
class EgonEhvSubstation (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus_id
    dbahn
    frequency
    lat
    lon
    operator
    osm_id
    osm_www
    point
    polygon
    power_type
    ref
    status
    subst_name
    substation
    voltage

class EgonHvmvSubstation (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus_id
```

```
dbahn
frequency
lat
lon
operator
osm_id
osm_www
point
polygon
power_type
ref
status
subst_name
substation
voltage

create_tables()
    Create tables for substation data :returns: None.

extract()
    Extract ehv and hvmv substation from transfer buses and results from osmtgmod

    Returns None.

class Osmtgmod(dependencies)
    Bases: egon.data.datasets.Dataset

import_osm_data()

osmtgmod(config_database='egon-data', config_basepath='osmTGmod/egon-data', con-
         fig_continue_run=False, filtered_osm_pbf_path_to_file=None, docker_db_config=None)

run()

to_pypsa()
```

10.5.49 power_etrageo

match_ocgt

Module containing the definition of the open cycle gas turbine links

insert_open_cycle_gas_turbines (*scn_name*='eGon2035')

Insert gas turbine links in egon_etrageo_link table.

Parameters *scn_name* (*str*) – Name of the scenario.

Returns *None*

map_buses (*scn_name*)

Map OCGT AC buses to nearest CH4 bus.

Parameters *scn_name* (*str*) – Name of the scenario.

Returns `gdf` (*geopandas.GeoDataFrame*) – GeoDataFrame with connected buses.

The central module containing all code dealing with ocgt in etrago

class `OpenCycleGasTurbineEtrago` (*dependencies*)

Bases: *egon.data.datasets.Dataset*

10.5.50 power_plants

assign_weather_data

find_bus_id (*power_plants, cfg*)

find_weather_id ()

Assign weather data to the weather dependant generators (wind and solar)

Parameters *No parameters required

weatherId_and_busId ()

write_power_plants_table (*power_plants, cfg, con*)

conventional

The module containing all code allocating power plants of different conventional technologies (oil, gas, others) based on data from MaStR and NEP.

match_nep_no_chp (*nep, mastr, matched, buffer_capacity=0.1, consider_location='plz', consider_carrier=True, consider_capacity=True*)

Match Power plants (no CHP) from MaStR to list of power plants from NEP

Parameters

- **nep** (*pandas.DataFrame*) – Power plants (no CHP) from NEP which are not matched to MaStR
- **mastr** (*pandas.DataFrame*) – Power plants (no CHP) from MaStR which are not matched to NEP
- **matched** (*pandas.DataFrame*) – Already matched power plants
- **buffer_capacity** (*float, optional*) – Maximum difference in capacity in p.u. The default is 0.1.

Returns

- **matched** (*pandas.DataFrame*) – Matched CHP
- **mastr** (*pandas.DataFrame*) – CHP plants from MaStR which are not matched to NEP
- **nep** (*pandas.DataFrame*) – CHP plants from NEP which are not matched to MaStR

select_nep_power_plants (*carrier*)

Select power plants with location from NEP's list of power plants

Parameters **carrier** (*str*) – Name of energy carrier

Returns *pandas.DataFrame* – Waste power plants from NEP list

select_no_chp_combustion_mastr (*carrier*)

Select power plants of a certain carrier from MaStR data which excludes all power plants used for allocation of CHP plants.

Parameters **carrier** (*str*) – Name of energy carrier

Returns *pandas.DataFrame* – Power plants from NEP list

mastr

Import MaStR dataset and write to DB tables

Data dump from Marktstammdatenregister (2022-11-17) is imported into the database. Only some technologies are taken into account and written to the following tables:

- PV: table *supply.egon_power_plants_pv*
- wind turbines: table *supply.egon_power_plants_wind*
- biomass/biogas plants: table *supply.egon_power_plants_biomass*
- hydro plants: table *supply.egon_power_plants_hydro*

Handling of empty source data in MaStr dump: * *voltage_level*: inferred based on nominal power (*capacity*) using the ranges from https://redmine.iks.cs.ovgu.de/oe/projects/ego-n/wiki/Definition_of_thresholds_for_voltage_level_assignment which results in True in column *voltage_level_inferred*. Remaining datasets are set to -1 (which only occurs if *capacity* is empty).

- *supply.egon_power_plants_*.bus_id*: set to -1 (only if not within grid districts or no geom available, e.g. for units with nom. power <30 kW)
- *supply.egon_power_plants_hydro.plant_type*: NaN

The data is used especially for the generation of status quo grids by ding0.

import_mastr () → None

Import MaStR data into database

infer_voltage_level (*units_gdf*: *geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*

Infer nan values in voltage level derived from generator capacity to the power plants.

Parameters

- **units_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing units with voltage levels from MaStR
- **Returns** *units_gdf* (*gpd.GeoDataFrame*)
- —
- **geopandas.GeoDataFrame** – *GeoDataFrame* containing units all having assigned a voltage level.

isfloat (*num*: *str*)

Determine if string can be converted to float. :Parameters: **num** (*str*) – String to parse.

Returns *bool* – Returns True in string can be parsed to float.

zip_and_municipality_from_standort (*standort*: *str*) → tuple[*str*, *bool*]

Get zip code and municipality from Standort string split into a list. :Parameters: **standort** (*str*) – Standort as given from MaStR data.

Returns *str* – Standort with only the zip code and municipality as well a ‘, Germany’ added.

pv_ground_mounted

insert()

pv_rooftop

The module containing all code dealing with pv rooftop distribution.

pv_rooftop_per_mv_grid()

Execute pv rooftop distribution method per scenario

Returns *None*.

pv_rooftop_per_mv_grid_and_scenario(scenario, level)

Interpolate solar rooftop per mv grid district

The target capacity is distributed to the mv grid districts linear to the residential and service electricity demands.

Parameters

- **scenario** (*str, optional*) – Name of the scenario
- **level** (*str, optional*) – Choose level of target values.

Returns *None*.

pv_rooftop_buildings

Distribute MaStR PV rooftop capacities to OSM and synthetic buildings. Generate new PV rooftop generators for scenarios eGon2035 and eGon100RE.

Data cleaning and inference: * Drop duplicates and entries with missing critical data. * Determine most plausible capacity from multiple values given in MaStR data. * Drop generators which don't have any plausible capacity data

($23.5\text{MW} > P > 0.1$).

- Randomly and weighted add a start-up date if it is missing.
- Extract zip and municipality from 'site' given in MaStR data.
- Geocode unique zip and municipality combinations with Nominatim (1 sec delay). Drop generators for which geocoding failed or which are located outside the municipalities of Germany.
- Add some visual sanity checks for cleaned data.

Allocation of MaStR data: * Allocate each generator to an existing building from OSM. * Determine the quantile each generator and building is in depending on the

capacity of the generator and the area of the polygon of the building.

- Randomly distribute generators within each municipality preferably within the same building area quantile as the generators are capacity wise.
- If not enough buildings exist within a municipality and quantile additional buildings from other quantiles are chosen randomly.

Desegregation of pv rooftop scenarios: * The scenario data per federal state is linearly distributed to the mv grid districts according to the pv rooftop potential per mv grid district.

- The rooftop potential is estimated from the building area given from the OSM buildings.

- Grid districts, which are located in several federal states, are allocated PV capacity according to their respective roof potential in the individual federal states.
- The desegregation of PV plants within a grid districts respects existing plants from MaStR, which did not reach their end of life.
- New PV plants are randomly and weighted generated using a breakdown of MaStR data as generator basis.
- Plant metadata (e.g. plant orientation) is also added random and weighted from MaStR data as basis.

```
class EgonPowerPlantPvRoofBuilding(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    building_id
    bus_id
    capacity
    gens_id
    index
    orientation_primary
    orientation_primary_angle
    orientation_uniform
    scenario
    voltage_level
    weather_cell_id
```

```
class OsmBuildingsFiltered(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    amenity
    area
    building
    geom
    geom_point
    id
    name
    osm_id
    tags
```

```
class Vg250Lan(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    ade
    ags
    ags_0
    ars
    ars_0
    bem
```



```

bez
bsg
debk_id
fk_s3
gen
geometry
gf
ibz
id
nbd
nuts
rs
rs_0
sdv_ars
sdv_rs
sn_g
sn_k
sn_l
sn_r
sn_v1
sn_v2
wsk

```

```

add_ags_to_buildings (buildings_gdf:      geopandas.geodataframe.GeoDataFrame,      munici-
                      palities_gdf:      geopandas.geodataframe.GeoDataFrame)      →      geopan-
                      das.geodataframe.GeoDataFrame

```

Add information about AGS ID to buildings. :Parameters: * **buildings_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing OSM buildings data.

- **municipalities_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame with municipality data.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing OSM buildings data with AGS ID added.

```

add_ags_to_gens (mastr_gdf:  geopandas.geodataframe.GeoDataFrame, municipalities_gdf:  geopan-
                    das.geodataframe.GeoDataFrame) → geopandas.geodataframe.GeoDataFrame

```

Add information about AGS ID to generators. :Parameters: * **mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame with valid and cleaned MaStR data.

- **municipalities_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame with municipality data.

Returns *geopandas.GeoDataFrame* – GeoDataFrame with valid and cleaned MaStR data with AGS ID added.

add_buildings_meta_data (*buildings_gdf*: *geopandas.geodataframe.GeoDataFrame*, *prob_dict*: *dict*, *seed*: *int*) → *geopandas.geodataframe.GeoDataFrame*
Randomly add additional metadata to disaggregated PV plants. :Parameters: * **buildings_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing OSM buildings data with disaggregated PV plants.

- **prob_dict** (*dict*) – Dictionary with values and probabilities per capacity range.
- **seed** (*int*) – Seed to use for random operations with NumPy and pandas.

Returns *geopandas.GeoDataFrame* – *GeoDataFrame* containing OSM building data with disaggregated PV plants.

add_bus_ids_sq (*buildings_gdf*: *geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*
Add bus ids for status_quo units

Parameters **buildings_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing OSM buildings data with disaggregated PV plants.

Returns *geopandas.GeoDataFrame* – *GeoDataFrame* containing OSM building data with bus_id per generator.

add_commissioning_date (*buildings_gdf*: *geopandas.geodataframe.GeoDataFrame*, *start*: *pandas._libs.tslibs.timestamps.Timestamp*, *end*: *pandas._libs.tslibs.timestamps.Timestamp*, *seed*: *int*)
Randomly and linear add start-up date to new pv generators. :Parameters: * **buildings_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing OSM buildings data with disaggregated PV plants.

- **start** (*pandas.Timestamp*) – Minimum Timestamp to use.
- **end** (*pandas.Timestamp*) – Maximum Timestamp to use.
- **seed** (*int*) – Seed to use for random operations with NumPy and pandas.

Returns *geopandas.GeoDataFrame* – *GeoDataFrame* containing OSM buildings data with start-up date added.

add_overlay_id_to_buildings (*buildings_gdf*: *geopandas.geodataframe.GeoDataFrame*, *grid_federal_state_gdf*: *geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*
Add information about overlay ID to buildings. :Parameters: * **buildings_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing OSM buildings data.

- **grid_federal_state_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* with intersection shapes between counties and grid districts.

Returns *geopandas.GeoDataFrame* – *GeoDataFrame* containing OSM buildings data with overlay ID added.

add_weather_cell_id (*buildings_gdf*: *geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*

allocate_pv (*q_mastr_gdf*: *gpd.GeoDataFrame*, *q_buildings_gdf*: *gpd.GeoDataFrame*, *seed*: *int*) → *tuple*[*gpd.GeoDataFrame*, *gpd.GeoDataFrame*]
Allocate the MaStR pv generators to the OSM buildings. This will determine a building for each pv generator if there are more buildings than generators within a given AGS. Primarily generators are distributed with

the same quantile as the buildings. Multiple assignment is excluded. :Parameters: * **q_mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing geocoded and qcut MaStR data.

- **q_buildings_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing qcut OSM buildings data.
- **seed** (*int*) – Seed to use for random operations with NumPy and pandas.

Returns *tuple with two geopandas.GeoDataFrame s* – GeoDataFrame containing MaStR data allocated to building IDs. GeoDataFrame containing building data allocated to MaStR IDs.

allocate_scenarios (*mastr_gdf: geopandas.geodataframe.GeoDataFrame, valid_buildings_gdf: geopandas.geodataframe.GeoDataFrame, last_scenario_gdf: geopandas.geodataframe.GeoDataFrame, scenario: str*)

Desaggregate and allocate scenario pv rooftop ramp-ups onto buildings. :Parameters: * **mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing geocoded MaStR data.

- **valid_buildings_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing OSM buildings data.
- **last_scenario_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing OSM buildings matched with pv generators from temporally preceding scenario.
- **scenario** (*str*) – Scenario to desaggrate and allocate.

Returns

tuple –

geopandas.GeoDataFrame GeoDataFrame containing OSM buildings matched with pv generators.

pandas.DataFrame DataFrame containing pv rooftop capacity per grid id.

allocate_to_buildings (*mastr_gdf: gpd.GeoDataFrame, buildings_gdf: gpd.GeoDataFrame*) → *tuple[gpd.GeoDataFrame, gpd.GeoDataFrame]*

Allocate status quo pv rooftop generators to buildings. :Parameters: * **mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing MaStR data with geocoded locations.

- **buildings_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing OSM buildings data with buildings without an AGS ID dropped.

Returns *tuple with two geopandas.GeoDataFrame s* – GeoDataFrame containing MaStR data allocated to building IDs. GeoDataFrame containing building data allocated to MaStR IDs.

building_area_range_per_cap_range (*mastr_gdf: gpd.GeoDataFrame, cap_ranges: list[tuple[int | float, int | float]] | None = None, min_building_size: int | float = 10.0, upper_quantile: float = 0.95, lower_quantile: float = 0.05*) → *dict[tuple[int | float, int | float], tuple[int | float, int | float]]*

Estimate normal building area range per capacity range. Calculate the mean roof load factor per capacity range from existing PV plants. :Parameters: * **mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing geocoded MaStR data.

- **cap_ranges** (*list(tuple(int, int))*) – List of capacity ranges to distinguish between. The first tuple should start with a zero and the last one should end with infinite.
- **min_building_size** (*int, float*) – Minimal building size to consider for PV plants.
- **upper_quantile** (*float*) – Upper quantile to estimate maximum building size per capacity range.
- **lower_quantile** (*float*) – Lower quantile to estimate minimum building size per capacity range.

Returns *dict* – Dictionary with estimated normal building area range per capacity range.

calculate_building_load_factor (*mastr_gdf*: *geopandas.geodataframe.GeoDataFrame*, *buildings_gdf*: *geopandas.geodataframe.GeoDataFrame*, *rounding*: *int* = 4) → *geopandas.geodataframe.GeoDataFrame*

Calculate the roof load factor from existing PV systems. :Parameters: * **mastr_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing geocoded MaStR data.

- **buildings_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing OSM buildings data.
- **rounding** (*int*) – Rounding to use for load factor.

Returns *geopandas.GeoDataFrame* – *GeoDataFrame* containing geocoded MaStR data with calculated load factor.

calculate_max_pv_cap_per_building (*buildings_gdf*: *gpd.GeoDataFrame*, *mastr_gdf*: *gpd.GeoDataFrame*, *pv_cap_per_sq_m*: *float* | *int*, *roof_factor*: *float* | *int*) → *gpd.GeoDataFrame*

Calculate the estimated maximum possible PV capacity per building.

Parameters

- **buildings_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing OSM buildings data.
- **mastr_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing geocoded MaStR data.
- **pv_cap_per_sq_m** (*float*, *int*) – Average expected, installable PV capacity per square meter.
- **roof_factor** (*float*, *int*) – Average for PV usable roof area share.

Returns *geopandas.GeoDataFrame* – *GeoDataFrame* containing OSM buildings data with estimated maximum PV capacity.

cap_per_bus_id (*scenario*: *str*) → *pandas.core.frame.DataFrame*

Get table with total pv rooftop capacity per grid district.

Parameters *scenario* (*str*) – Scenario name.

Returns *pandas.DataFrame* – *DataFrame* with total rooftop capacity per mv grid.

cap_share_per_cap_range (*mastr_gdf*: *gpd.GeoDataFrame*, *cap_ranges*: *list[tuple[int | float, int | float]]* | *None* = *None*) → *dict[tuple[int | float, int | float], float]*

Calculate the share of PV capacity from the total PV capacity within capacity ranges.

Parameters

- **mastr_gdf** (*geopandas.GeoDataFrame*) – *GeoDataFrame* containing geocoded MaStR data.
- **cap_ranges** (*list[tuple(int, int)]*) – List of capacity ranges to distinguish between. The first tuple should start with a zero and the last one should end with infinite.

Returns *dict* – Dictionary with share of PV capacity from the total PV capacity within capacity ranges.

clean_mastr_data (*mastr_gdf*: *gpd.GeoDataFrame*, *max_realistic_pv_cap*: *int* | *float*, *min_realistic_pv_cap*: *int* | *float*, *seed*: *int*) → *gpd.GeoDataFrame*

Clean the MaStR data from implausible data.

- Drop MaStR ID duplicates.
- Drop generators with implausible capacities.

Parameters

- **mastr_gdf** (*pandas.DataFrame*) – DataFrame containing MaStR data.
- **max_realistic_pv_cap** (*int or float*) – Maximum capacity, which is considered to be realistic.
- **min_realistic_pv_cap** (*int or float*) – Minimum capacity, which is considered to be realistic.
- **seed** (*int*) – Seed to use for random operations with NumPy and pandas.

Returns *pandas.DataFrame* – DataFrame containing cleaned MaStR data.

create_scenario_table (*buildings_gdf*)

Create mapping table pv_unit <-> building for scenario

desaggregate_pv (*buildings_gdf: geopandas.geodataframe.GeoDataFrame, cap_df: pandas.core.frame.DataFrame, **kwargs*) → *geopandas.geodataframe.GeoDataFrame*

Desaggregate PV capacity on buildings within a given grid district.

Parameters

- **buildings_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing OSM buildings data.
- **cap_df** (*pandas.DataFrame*) – DataFrame with total rooftop capacity per mv grid.

Other Parameters

- **prob_dict** (*dict*) – Dictionary with values and probabilities per capacity range.
- **cap_share_dict** (*dict*) – Dictionary with share of PV capacity from the total PV capacity within capacity ranges.
- **building_area_range_dict** (*dict*) – Dictionary with estimated normal building area range per capacity range.
- **load_factor_dict** (*dict*) – Dictionary with mean roof load factor per capacity range.
- **seed** (*int*) – Seed to use for random operations with NumPy and pandas.
- **pv_cap_per_sq_m** (*float, int*) – Average expected, installable PV capacity per square meter.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing OSM building data with desaggregated PV plants.

desaggregate_pv_in_mv_grid (*buildings_gdf: gpd.GeoDataFrame, pv_cap: float | int, **kwargs*) → *gpd.GeoDataFrame*

Desaggregate PV capacity on buildings within a given grid district. :Parameters: * **buildings_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing buildings within the grid district.

- **pv_cap** (*float, int*) – PV capacity to desaggregate.

Other Parameters

- **prob_dict** (*dict*) – Dictionary with values and probabilities per capacity range.
- **cap_share_dict** (*dict*) – Dictionary with share of PV capacity from the total PV capacity within capacity ranges.
- **building_area_range_dict** (*dict*) – Dictionary with estimated normal building area range per capacity range.
- **load_factor_dict** (*dict*) – Dictionary with mean roof load factor per capacity range.
- **seed** (*int*) – Seed to use for random operations with NumPy and pandas.

- **pv_cap_per_sq_m** (*float, int*) – Average expected, installable PV capacity per square meter.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing OSM building data with disaggregated PV plants.

determine_end_of_life_gens (*mastr_gdf: geopandas.geodataframe.GeoDataFrame, scenario_timestamp: pandas._libs.tslibs.timestamps.Timestamp, pv_rooftop_lifetime: pandas._libs.tslibs.timedeltas.Timedelta*) → *geopandas.geodataframe.GeoDataFrame*

Determine if an old PV system has reached its end of life. :Parameters: * **mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing geocoded MaStR data.

- **scenario_timestamp** (*pandas.Timestamp*) – Timestamp at which the scenario takes place.
- **pv_rooftop_lifetime** (*pandas.Timedelta*) – Average expected lifetime of PV rooftop systems.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing geocoded MaStR data and info if the system has reached its end of life.

drop_buildings_outside_grids (*buildings_gdf: geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*

Drop all buildings outside of grid areas. :Parameters: **buildings_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing OSM buildings data.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing OSM buildings data with buildings without an bus ID dropped.

drop_buildings_outside_muns (*buildings_gdf: geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*

Drop all buildings outside of municipalities. :Parameters: **buildings_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing OSM buildings data.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing OSM buildings data with buildings without an AGS ID dropped.

drop_gens_outside_muns (*mastr_gdf: geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*

Drop all generators outside of municipalities. :Parameters: **mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame with valid and cleaned MaStR data.

Returns *geopandas.GeoDataFrame* – GeoDataFrame with valid and cleaned MaStR data with generators without an AGS ID dropped.

drop_unallocated_gens (*gdf: geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*

Drop generators which did not get allocated.

Parameters **gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing MaStR data allocated to building IDs.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing MaStR data with generators dropped which did not get allocated.

egon_building_peak_loads ()

federal_state_data (*to_crs: pyproj.crs.crs.CRS*) → *geopandas.geodataframe.GeoDataFrame*

Get feder state data from eGo^N Database. :Parameters: **to_crs** (*pyproj.crs.crs.CRS*) – CRS to transform geometries to.

Returns *geopandas.GeoDataFrame* – GeoDataFrame with federal state data.

frame_to_numeric (*df: pd.DataFrame | gpd.GeoDataFrame*) → *pd.DataFrame | gpd.GeoDataFrame*

Try to convert all columns of a DataFrame to numeric ignoring errors. :Parameters: **df** (*pandas.DataFrame or geopandas.GeoDataFrame*)

Returns *pandas.DataFrame or geopandas.GeoDataFrame*

get_probability_for_property (*mastr_gdf: gpd.GeoDataFrame, cap_range: tuple[int | float, int | float], prop: str*) → *tuple[np.array, np.array]*

Calculate the probability of the different options of a property of the existing PV plants. :Parameters: * **mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing geocoded MaStR data.

- **cap_range** (*tuple(int, int)*) – Capacity range of PV plants to look at.
- **prop** (*str*) – Property to calculate probabilities for. String needs to be in columns of *mastr_gdf*.

Returns

tuple –

numpy.array Unique values of property.

numpy.array Probabilities per unique value.

grid_districts (*epsg: int*) → *geopandas.geodataframe.GeoDataFrame*

Load mv grid district geo data from eGo^N Database as *geopandas.GeoDataFrame*. :Parameters: **epsg** (*int*) – EPSG ID to use as CRS.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing mv grid district ID and geo shapes data.

infer_voltage_level (*units_gdf: geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*

Infer nan values in voltage level derived from generator capacity to the power plants.

Parameters

- **units_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing units with voltage levels from MaStR
- **Returns** *units_gdf* (*gpd.GeoDataFrame*)
- ———
- **geopandas.GeoDataFrame** – GeoDataFrame containing units all having assigned a voltage level.

load_building_data ()

Read buildings from DB Tables:

- *openstreetmap.osm_buildings_filtered* (from OSM)
- *openstreetmap.osm_buildings_synthetic* (synthetic, created by us)

Use column *id* for both as it is unique hence you concat both datasets. If **INCLUDE_SYNTHETIC_BUILDINGS** is False synthetic buildings will not be loaded.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing OSM buildings data with buildings without an AGS ID dropped.

load_mastr_data ()

Read PV rooftop data from MaStR CSV Note: the source will be replaced as soon as the MaStR data is available in DB. :returns: *geopandas.GeoDataFrame* – GeoDataFrame containing MaStR data with geocoded locations.

mastr_data (*index_col: str | int | list[str] | list[int]*) → *gpd.GeoDataFrame*

Read MaStR data from database.

Parameters `index_col` (*str, int or list of str or int*) – Column(s) to use as the row labels of the DataFrame.

Returns `pandas.DataFrame` – DataFrame containing MaStR data.

mean_load_factor_per_cap_range (*mastr_gdf: gpd.GeoDataFrame, cap_ranges: list[tuple[int | float, int | float]] | None = None*) → dict[tuple[int | float, int | float], float]

Calculate the mean roof load factor per capacity range from existing PV plants. :Parameters: * **mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing geocoded MaStR data.

- **cap_ranges** (*list(tuple(int, int))*) – List of capacity ranges to distinguish between. The first tuple should start with a zero and the last one should end with infinite.

Returns *dict* – Dictionary with mean roof load factor per capacity range.

municipality_data () → `geopandas.geodataframe.GeoDataFrame`

Get municipality data from eGo^N Database. :returns: *geopandas.GeoDataFrame* – GeoDataFrame with municipality data.

osm_buildings (*to_crs: pyproj.crs.crs.CRS*) → `geopandas.geodataframe.GeoDataFrame`

Read OSM buildings data from eGo^N Database. :Parameters: **to_crs** (*pyproj.crs.crs.CRS*) – CRS to transform geometries to.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing OSM buildings data.

overlay_grid_districts_with_counties (*mv_grid_district_gdf: geopandas.geodataframe.GeoDataFrame, federal_state_gdf: geopandas.geodataframe.GeoDataFrame*) → *geopandas.geodataframe.GeoDataFrame*

Calculate the intersections of mv grid districts and counties. :Parameters: * **mv_grid_district_gdf** (*gpd.GeoDataFrame*) – GeoDataFrame containing mv grid district ID and geo shapes data.

- **federal_state_gdf** (*gpd.GeoDataFrame*) – GeoDataFrame with federal state data.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing OSM buildings data.

probabilities (*mastr_gdf: gpd.GeoDataFrame, cap_ranges: list[tuple[int | float, int | float]] | None = None, properties: list[str] | None = None*) → dict

Calculate the probability of the different options of properties of the existing PV plants. :Parameters: * **mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing geocoded MaStR data.

- **cap_ranges** (*list(tuple(int, int))*) – List of capacity ranges to distinguish between. The first tuple should start with a zero and the last one should end with infinite.
- **properties** (*list(str)*) – List of properties to calculate probabilities for. Strings need to be in columns of `mastr_gdf`.

Returns *dict* – Dictionary with values and probabilities per capacity range.

pv_rooftop_to_buildings ()

Main script, executed as task

scenario_data (*carrier: str = 'solar_rooftop', scenario: str = 'eGon2035'*) → *pandas.core.frame.DataFrame*

Get scenario capacity data from eGo^N Database. :Parameters: * **carrier** (*str*) – Carrier type to filter table by.

- **scenario** (*str*) – Scenario to filter table by.

Returns *geopandas.GeoDataFrame* – GeoDataFrame with scenario capacity data in GW.

sort_and_qcut_df (*df*: *pd.DataFrame* | *gpd.GeoDataFrame*, *col*: *str*, *q*: *int*) → *pd.DataFrame* | *gpd.GeoDataFrame*

Determine the quantile of a given attribute in a (Geo)DataFrame. Sort the (Geo)DataFrame in ascending order for the given attribute. :Parameters: * **df** (*pandas.DataFrame* or *geopandas.GeoDataFrame*) – (Geo)DataFrame to sort and qcut.

- **col** (*str*) – Name of the attribute to sort and qcut the (Geo)DataFrame on.
- **q** (*int*) – Number of quantiles.

Returns *pandas.DataFrame* or *geopandas.GeoDataFrame* – Sorted and qcut (Geo)DataFrame.

synthetic_buildings (*to_crs*: *pyproj.crs.crs.CRS*) → *geopandas.geodataframe.GeoDataFrame*

Read synthetic buildings data from eGoⁿ Database. :Parameters: **to_crs** (*pyproj.crs.crs.CRS*) – CRS to transform geometries to.

Returns *geopandas.GeoDataFrame* – GeoDataFrame containing OSM buildings data.

timer_func (*func*)

validate_output (*desagg_mastr_gdf*: *pd.DataFrame* | *gpd.GeoDataFrame*, *desagg_buildings_gdf*: *pd.DataFrame* | *gpd.GeoDataFrame*) → *None*

Validate output.

- Validate that there are exactly as many buildings with a pv system as there are pv systems with a building
- Validate that the building IDs with a pv system are the same building IDs as assigned to the pv systems
- Validate that the pv system IDs with a building are the same pv system IDs as assigned to the buildings

Parameters

- **desagg_mastr_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing MaStR data allocated to building IDs.
- **desagg_buildings_gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame containing building data allocated to MaStR IDs.

wind_farms

generate_map ()

Generates a map with the position of all the wind farms

Parameters *No parameters required

generate_wind_farms ()

Generate wind farms based on existing wind farms.

Parameters *No parameters required

insert ()

Main function. Import power objectives generate results calling the functions “generate_wind_farms” and “wind_power_states”.

Parameters *No parameters required

wind_power_states (*state_wf*, *state_wf_ni*, *state_mv_districts*, *target_power*, *scenario_year*, *source*, *fed_state*)

Import OSM data from a Geofabrik .pbx file into a PostgreSQL database.

Parameters

- **state_wf** (*geodataframe, mandatory*) – gdf containing all the wf in the state created based on existing wf.
- **state_wf_ni** (*geodataframe, mandatory*) – potential areas in the the state wich don't intersect any existing wf
- **state_mv_districts** (*geodataframe, mandatory*) – gdf containing all the MV/HV substations in the state
- **target_power** (*int, mandatory*) – Objective power for a state given in MW
- **scenario_year** (*str, mandatory*) – name of the scenario
- **source** (*str, mandatory*) – Type of energy genetor. Always “Wind_onshore” for this script.
- **fed_state** (*str, mandatory*) – Name of the state where the wind farms will be allocated

wind_offshore

insert()

Include the offshore wind parks in egon-data. locations and installed capacities based on: NEP2035_V2021_scnC2035

Parameters *No parameters required

The central module containing all code dealing with power plant data.

class EgonPowerPlants (***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`

bus_id

carrier

el_capacity

geom

id

scenario

source_id

sources

voltage_level

weather_cell_id

class PowerPlants (*dependencies*)

Bases: `egon.data.datasets.Dataset`

This module creates all electrical generators for different scenarios. It also calculates the weather area for each weather dependent generator.

Dependencies

- *Chp*
- `:py:class:‘CtsElectricityDemand`

```
<egon.data.datasets.electricity_demand.CtsElectricityDemand>‘ * HouseholdElectricityDemand
* mastr_data * define_mv_grid_districts * RePotentialAreas * ZensusVg250 *
ScenarioCapacities * ScenarioParameters * Setup * substation_extraction *
Vg250MvGridDistricts * ZensusMvGridDistricts
```

Resulting tables

- :py:class:‘supply.egon_power_plants

<egon.data.datasets.power_plants.EgonPowerPlants>‘ is filled

name = 'PowerPlants'

version = '0.0.18'

allocate_conventional_non_chp_power_plants ()

allocate_other_power_plants ()

assign_bus_id (power_plants, cfg)

Assigns bus_ids to power plants according to location and voltage level

Parameters power_plants (pandas.DataFrame) – Power plants including voltage level

Returns power_plants (pandas.DataFrame) – Power plants including voltage level and bus_id

assign_voltage_level (mastr_loc, cfg, mastr_working_dir)

Assigns voltage level to power plants.

If location data including voltage level is available from Marktstammdatenregister, this is used. Otherwise the voltage level is assigned according to the electrical capacity.

Parameters mastr_loc (pandas.DataFrame) – Power plants listed in MaStR with geometry inside German boundaries

Returns pandas.DataFrame – Power plants including voltage_level

assign_voltage_level_by_capacity (mastr_loc)

create_tables ()

Create tables for power plant data :returns: None.

filter_mastr_geometry (mastr, federal_state=None)

Filter data from MaStR by geometry

Parameters

- **mastr** (pandas.DataFrame) – All power plants listed in MaStR
- **federal_state** (str or None) – Name of federal state whose power plants are returned. If None, data for Germany is returned

Returns mastr_loc (pandas.DataFrame) – Power plants listed in MaStR with geometry inside German boundaries

insert_biomass_plants (scenario)

Insert biomass power plants of future scenario

Parameters scenario (str) – Name of scenario.

Returns None.

insert_hydro_biomass ()

Insert hydro and biomass power plants in database

Returns None.

insert_hydro_plants (scenario)

Insert hydro power plants of future scenario.

Hydro power plants are divided into run_of_river and reservoir plants according to Marktstammdatenregister. Additional hydro technologies (e.g. turbines inside drinking water systems) are not considered.

Parameters `scenario` (*str*) – Name of scenario.

Returns *None*.

scale_prox2now (*df*, *target*, *level*='federal_state')

Scale installed capacities linear to status quo power plants

Parameters

- **df** (*pandas.DataFrame*) – Status Quo power plants
- **target** (*pandas.Series*) – Target values for future scenario
- **level** (*str*, *optional*) – Scale per 'federal_state' or 'country'. The default is 'federal_state'.

Returns *df* (*pandas.DataFrame*) – Future power plants

select_target (*carrier*, *scenario*)

Select installed capacity per scenario and carrier

Parameters

- **carrier** (*str*) – Name of energy carrier
- **scenario** (*str*) – Name of scenario

Returns *pandas.Series* – Target values for carrier and scenario

10.5.51 pypsaEurSec

The central module containing all code dealing with importing data from the pypsa-eur-sec scenario parameter creation

class PypsaEurSec (*dependencies*)

Bases: *egon.data.datasets.Dataset*

clean_database ()

Remove all components abroad for eGon100RE of the database

Remove all components abroad and their associated time series of the dataset for the scenario 'eGon100RE'.

Parameters *None*

Returns *None*

neighbor_reduction ()

overwrite_H2_pipeline_share ()

Overwrite retrofitted_CH4pipeline-to-H2pipeline_share value

Overwrite retrofitted_CH4pipeline-to-H2pipeline_share in the scenario parameter table if p-e-s is run. This function writes in the database and has no return.

read_network ()

run_pypsa_eur_sec ()

10.5.52 re_potential_areas

The central module containing all code dealing with importing data on potential areas for wind onshore and ground-mounted PV.

class EgonRePotentialAreaPvAgriculture (***kwargs*)

Bases: *sqlalchemy.ext.declarative.api.Base*

```

    geom
    id
class EgonRePotentialAreaPvRoadRailway (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    geom
    id
class EgonRePotentialAreaWind (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    geom
    id
create_tables ()
    Create tables for RE potential areas
insert_data ()
    Insert data into DB

```

10.5.53 saltcavern

The central module containing all code dealing with bgr data.

This module either directly contains the code dealing with importing bgr data, or it re-exports everything needed to handle it. Please refrain from importing code from any modules below this one, because it might lead to unwanted behaviour.

If you have to import code from a module below this one because the code isn't exported from this module, please file a bug, so we can fix this.

```

class SaltcavernData (dependencies)
    Bases: egon.data.datasets.Dataset
to_postgres ()
    Write BGR saline structures to database.

```

10.5.54 scenario_parameters

parameters

The module containing all parameters for the scenario table

annualize_capital_costs (*overnight_costs, lifetime, p*)

Parameters

- **overnight_costs** (*float*) – Overnight investment costs in EUR/MW or EUR/MW/km
- **lifetime** (*int*) – Number of years in which payments will be made
- **p** (*float*) – Interest rate in p.u.

Returns *float* – Annualized capital costs in EUR/MW/a or EUR/MW/km/a

electricity (*scenario*)

Returns paramaters of the electricity sector for the selected scenario.

Parameters `scenario` (*str*) – Name of the scenario.

Returns `parameters` (*dict*) – List of parameters of electricity sector

gas (*scenario*)

Returns parameters of the gas sector for the selected scenario.

Parameters `scenario` (*str*) – Name of the scenario.

Returns `parameters` (*dict*) – List of parameters of gas sector

global_settings (*scenario*)

Returns global parameters for the selected scenario.

Parameters `scenario` (*str*) – Name of the scenario.

Returns `parameters` (*dict*) – List of global parameters

heat (*scenario*)

Returns parameters of the heat sector for the selected scenario.

Parameters `scenario` (*str*) – Name of the scenario.

Returns `parameters` (*dict*) – List of parameters of heat sector

mobility (*scenario*)

Returns parameters of the mobility sector for the selected scenario.

Parameters `scenario` (*str*) – Name of the scenario.

Returns `parameters` (*dict*) – List of parameters of mobility sector

Notes

For a detailed description of the parameters see module `egon.data.datasets.emobility.motorized_individual_travel`.

read_costs (*df, technology, parameter, value_only=True*)

read_csv (*year*)

The central module containing all code dealing with scenario table.

class `EgonScenario` (***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`

description

electricity_parameters

gas_parameters

global_parameters

heat_parameters

mobility_parameters

name

class `ScenarioParameters` (*dependencies*)

Bases: `egon.data.datasets.Dataset`

Create and fill table with central parameters for each scenario

This dataset creates and fills a table in the database that includes central parameters for each scenarios. These parameters are mostly from external sources, they are defined and referenced within this dataset. The table is accessed by various datasets to access the parameters for all sectors.

Dependencies

- *Setup*

Resulting tables

- *scenario.egon_scenario_parameters* is created and filled

name = 'ScenarioParameters'

version = '0.0.12'

create_table()

Create table for scenarios :returns: *None*.

download_pypsa_technology_data()

Download PyPSA technology data results.

get_sector_parameters (*sector, scenario=None*)

Returns parameters for each sector as dictionary.

If *scenario=None* data for all scenarios is returned as *pandas.DataFrame*. Otherwise the parameters of the specific scenario are returned as a dict.

Parameters

- **sector** (*str*) – Name of the sector. Options are: ['global', 'electricity', 'heat', 'gas', 'mobility']
- **scenario** (*str, optional*) – Name of the scenario. The default is *None*.

Returns values (*dict or pandas.DataFrame*) – List or table of parameters for the selected sector

insert_scenarios()

Insert scenarios and their parameters to scenario table

Returns *None*.

10.5.55 storages

home_batteries

Home Battery allocation to buildings

Main module for allocation of home batteries onto buildings and sizing them depending on pv rooftop system size.

Contents of this module * Creation of DB tables * Allocate given home battery capacity per mv grid to buildings with pv rooftop

systems. The sizing of the home battery system depends on the size of the pv rooftop system and can be set within the *datasets.yml*. Default sizing is 1:1 between the pv rooftop capacity (kWp) and the battery capacity (kWh).

- Write results to DB

Configuration

The config of this dataset can be found in *datasets.yml* in section *home_batteries*.

Scenarios and variations

Assumptions can be changed within the *datasets.yml*.

Only buildings with a pv rooftop systems are considered within the allocation process. The default sizing of home batteries is 1:1 between the pv rooftop capacity (kWp) and the battery capacity (kWh). Reaching the exact value of the allocation of battery capacities per grid area leads to slight deviations from this specification.

Methodology

The selection of buildings is done randomly until a result is reached which is close to achieving the sizing specification.

```
class EgonHomeBatteries (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    building_id
    bus_id
    capacity
    index
    p_nom
    scenario

    targets = {'home_batteries': {'schema': 'supply', 'table': 'egon_home_batteries'}}
```

allocate_home_batteries_to_buildings()
Allocate home battery storage systems to buildings with pv rooftop systems

create_table(df)
Create mapping table home battery <-> building id

get_cbat_pbat_ratio()
Mean ratio between the storage capacity and the power of the pv rooftop system

Returns *int* – Mean ratio between the storage capacity and the power of the pv rooftop system

pumped_hydro

The module containing code allocating pumped hydro plants based on data from MaStR and NEP.

apply_voltage_level_thresholds(power_plants)
Assigns voltage level to power plants based on thresholds defined for the egon project.

Parameters *power_plants* (*pandas.DataFrame*) – Power plants and their electrical capacity

Returns *pandas.DataFrame* – Power plants including voltage_level

get_location(unmatched)
Gets a geolocation for units which couldn't be matched using MaStR data. Uses geolocator and the city name from NEP data to create longitude and latitude for a list of unmatched units.

Parameters *unmatched* (*pandas.DataFrame*) – storage units from NEP which are not matched to MaStR but containing a city information

Returns

- **unmatched** (*pandas.DataFrame*) – Units for which no geolocation could be identified
- **located** (*pandas.DataFrame*) – Units with a geolocation based on their city information

match_storage_units (*nep, mastr, matched, buffer_capacity=0.1, consider_location='plz', consider_carrier=True, consider_capacity=True*)

Match storage_units (in this case only pumped hydro) from MaStR to list of power plants from NEP

Parameters

- **nep** (*pandas.DataFrame*) – storage units from NEP which are not matched to MaStR
- **mastr** (*pandas.DataFrame*) – Pstorage_units from MaStR which are not matched to NEP
- **matched** (*pandas.DataFrame*) – Already matched storage_units
- **buffer_capacity** (*float, optional*) – Maximum difference in capacity in p.u. The default is 0.1.

Returns

- **matched** (*pandas.DataFrame*) – Matched CHP
- **mastr** (*pandas.DataFrame*) – storage_units from MaStR which are not matched to NEP
- **nep** (*pandas.DataFrame*) – storage_units from NEP which are not matched to MaStR

select_mastr_pumped_hydro ()

Select pumped hydro plants from MaStR

Returns *pandas.DataFrame* – Pumped hydro plants from MaStR

select_nep_pumped_hydro ()

Select pumped hydro plants from NEP power plants list

Returns *pandas.DataFrame* – Pumped hydro plants from NEP list

The central module containing all code dealing with power plant data.

class EgonStorages (***kwargs*)

Bases: *sqlalchemy.ext.declarative.api.Base*

bus_id

carrier

el_capacity

geom

id

scenario

source_id

sources

voltage_level

class Storages (*dependencies*)

Bases: *egon.data.datasets.Dataset*

allocate_pumped_hydro_eGon100RE ()

Allocates pumped_hydro plants for eGon100RE scenario based on a prox-to-now method applied on allocated pumped-hydro plants in the eGon2035 scenario.

Parameters *None*

Returns *None*

allocate_pumped_hydro_eGon2035 (*export=True*)

Allocates pumped_hydro plants for eGon2035 scenario and either exports results to data base or returns as a dataframe

Parameters **export** (*bool*) – Choose if allocated pumped hydro plants should be exported to the data base. The default is True. If export=False a data frame will be returned

Returns **power_plants** (*pandas.DataFrame*) – List of pumped hydro plants in ‘eGon2035’ scenario

allocate_pv_home_batteries_to_grids ()

create_tables ()

Create tables for power plant data :returns: *None*.

home_batteries_per_scenario (*scenario*)

Allocates home batteries which define a lower boundary for extendable battery storage units. The overall installed capacity is taken from NEP for eGon2035 scenario. The spatial distribution of installed battery capacities is based on the installed pv rooftop capacity.

Parameters **None**

Returns *None*

10.5.56 storages_etrageo

The central module containing all code dealing with existing storage units for eTraGo.

class StorageEtrageo (*dependencies*)

Bases: *egon.data.datasets.Dataset*

extendable_batteries ()

extendable_batteries_per_scenario (*scenario*)

insert_PHES ()

10.5.57 substation

The central module containing code to create substation tables

class EgonEhvTransferBuses (***kwargs*)

Bases: *sqlalchemy.ext.declarative.api.Base*

bus_id

dbahn

frequency

lat

lon

operator

osm_id

osm_www

point

polygon

power_type

```

    ref
    status
    subst_name
    substation
    voltage

class EgonHvmvTransferBuses (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    bus_id
    dbahn
    frequency
    lat
    lon
    operator
    osm_id
    osm_www
    point
    polygon
    power_type
    ref
    status
    subst_name
    substation
    voltage

class SubstationExtraction (dependencies)
    Bases: egon.data.datasets.Dataset

create_sql_functions ()
    Defines Postgresql functions needed to extract substation from osm

    Returns None.

create_tables ()
    Create tables for substation data :returns: None.

transfer_busses ()

```

10.5.58 vg250

The central module containing all code dealing with VG250 data.

This module either directly contains the code dealing with importing VG250 data, or it re-exports everything needed to handle it. Please refrain from importing code from any modules below this one, because it might lead to unwanted behaviour.

If you have to import code from a module below this one because the code isn't exported from this module, please file a bug, so we can fix this.

class **Vg250** (*dependencies*)

Bases: *egon.data.datasets.Dataset*

filename = 'https://daten.gdz.bkg.bund.de/produkte/vg/vg250_ebenen_0101/2020/vg250_01-

add_metadata ()

Writes metadata JSON string into table comment.

cleaning_and_preperation ()

download_files ()

Download VG250 (Verwaltungsgebiete) shape files.

nuts_mview ()

to_postgres ()

vg250_metadata_resources_fields ()

10.5.59 zensus

The central module containing all code dealing with importing Zensus data.

class **ZensusMiscellaneous** (*dependencies*)

Bases: *egon.data.datasets.Dataset*

class **ZensusPopulation** (*dependencies*)

Bases: *egon.data.datasets.Dataset*

adjust_zensus_misc ()

Delete unpopulated cells in zensus-households, -buildings and -apartments

Some unpopulated zensus cells are listed in: - *egon_destatis_zensus_household_per_ha* -
egon_destatis_zensus_building_per_ha - *egon_destatis_zensus_apartment_per_ha*

This can be caused by missing population information due to privacy or other special cases (e.g. holiday homes are listed as buildings but are not permanently populated.) In the following tasks of *egon-data*, only data of populated cells is used.

Returns *None*.

create_combined_zensus_table ()

Create combined table with buildings, apartments and population per cell

Only apartment and building data with acceptable data quality (*quantity_q*<2) is used, all other data is dropped. For more details on data quality see Zensus docs: <https://www.zensus2011.de/DE/Home/Aktuelles/DemografischeGrunddaten.html>

If there's no data on buildings or apartments for a certain cell, the value for *building_count* resp. *apartment_count* contains NULL.

create_zensus_misc_tables ()

Create tables for zensus data in postgres database

create_zensus_pop_table ()

Create tables for zensus data in postgres database

download_and_check (*url*, *target_file*, *max_iteration*=5)

Download file from url (http) if it doesn't exist and check afterwards. If bad zip remove file and re-download. Repeat until file is fine or reached maximum iterations.

download_zensus_misc()

Download Zensus csv files on data per hectare grid cell.

download_zensus_pop()

Download Zensus csv file on population per hectare grid cell.

filter_zensus_misc(*filename, dataset*)

This block filters lines in the source CSV file and copies the appropriate ones to the destination based on grid_id values.

Parameters

- **filename** (*str*) – Path to input csv-file
- **dataset** (*str, optional*) – Toggles between production (*dataset='Everything'*) and test mode e.g. (*dataset='Schleswig-Holstein'*). In production mode, data covering entire Germany is used. In the test mode a subset of this data is used for testing the workflow.

Returns *str* – Path to output csv-file

filter_zensus_population(*filename, dataset*)

This block filters lines in the source CSV file and copies the appropriate ones to the destination based on geometry.

Parameters

- **filename** (*str*) – Path to input csv-file
- **dataset** (*str, optional*) – Toggles between production (*dataset='Everything'*) and test mode e.g. (*dataset='Schleswig-Holstein'*). In production mode, data covering entire Germany is used. In the test mode a subset of this data is used for testing the workflow.

Returns *str* – Path to output csv-file

population_to_postgres()

Import Zensus population data to postgres database

select_geom()

Select the union of the geometries of Schleswig-Holstein from the database, convert their projection to the one used in the CSV file, output the result to stdout as a GeoJSON string and read it into a prepared shape for filtering.

target (*source, dataset*)

Generate the target path corresponding to a source path.

Parameters **dataset** (*str*) – Toggles between production (*dataset='Everything'*) and test mode e.g. (*dataset='Schleswig-Holstein'*). In production mode, data covering entire Germany is used. In the test mode a subset of this data is used for testing the workflow.

Returns *Path* – Path to target csv-file

zensus_misc_to_postgres()

Import data on buildings, households and apartments to postgres db

The API for configuring datasets.

class Dataset (*name: 'str', version: 'str', dependencies: 'Iterable[Union[Dataset, Task]]' = (), tasks: 'Union[Tasks, TaskGraph]' = ()*)

Bases: object

check_version (*after_execution=()*)

dependencies = ()

The first task(s) of this *Dataset* will be marked as downstream of any of the listed dependencies. In case

of bare *Task*, a direct link will be created whereas for a *Dataset* the link will be made to all of its last tasks.

name = None

The name of the Dataset

tasks = ()

The tasks of this *Dataset*. A *TaskGraph* will automatically be converted to *Tasks*.

update (*session*)

version = None

The *Dataset*'s version. Can be anything from a simple semantic versioning string like "2.1.3", to a more complex string, like for example "2021-01-01.schleswig-holstein.0" for OpenStreetMap data. Note that the latter encodes the *Dataset*'s date, region and a sequential number in case the data changes without the date or region changing, for example due to implementation changes.

class Model (**kwargs)

Bases: sqlalchemy.ext.declarative.api.Base

dependencies

epoch

id

name

version

Task = typing.Union[typing.Callable[[], NoneType], airflow.models.baseoperator.BaseOperator]

A *Task* is an Airflow Operator or any Callable taking no arguments and returning None. Callables will be converted to Operators by wrapping them in a *PythonOperator* and setting the *task_id* to the Callable's *__name__*, with underscores replaced with hyphens. If the Callable's *__module__* attribute contains the string "egon.data.datasets.", the *task_id* is also prefixed with the module name, followed by a dot and with "egon.data.datasets." removed.

TaskGraph = typing.Union[typing.Callable[[], NoneType], airflow.models.baseoperator.BaseOperator]

A graph of tasks is, in its simplest form, just a single node, i.e. a single *Task*. More complex graphs can be specified by nesting sets and tuples of *TaskGraphs*. A set of *TaskGraphs* means that they are unordered and can be executed in parallel. A tuple specifies an implicit ordering so a tuple of *TaskGraphs* will be executed sequentially in the given order.

class Tasks (*graph: 'TaskGraph'*)

Bases: dict

graph = ()

prefix (*o*)

setup ()

Create the database structure for storing dataset information.

10.6 db

assign_gas_bus_id (*dataframe, scn_name, carrier*)

Assign 'bus_id's to points according to location.

The points are taken from the given *dataframe* and the geometries by which the *bus_id's* are assigned to them are taken from the 'grid.egon_gas_voronoi' table.

Parameters

- **dataframe** (*pandas.DataFrame*) – DataFrame containing points
- **scn_name** (*str*) – Name of the scenario
- **carrier** (*str*) – Name of the carrier

Returns *res* (*pandas.DataFrame*) – Dataframe including bus_id

check_db_unique_violation (*func*)

Wrapper to catch psycopg's UniqueViolation errors during concurrent DB commits.

Preferably used with *next_etrango_id()*. Retries DB operation 10 times before raising original exception.

Can be used as a decorator like this:

```
>>> @check_db_unique_violation
... def commit_something_to_database():
...     # commit something here
...     return
...
>>> commit_something_to_database() # doctest: +SKIP
```

Examples

Add new bus to eTraGo's bus table:

```
>>> from egon.data import db
>>> from egon.data.datasets.etrango_setup import EgonPfHvBus
...
>>> @check_db_unique_violation
... def add_etrango_bus():
...     bus_id = db.next_etrango_id("bus")
...     with db.session_scope() as session:
...         emob_bus_id = db.next_etrango_id("bus")
...         session.add(
...             EgonPfHvBus(
...                 scn_name="eGon2035",
...                 bus_id=bus_id,
...                 v_nom=1,
...                 carrier="whatever",
...                 x=52,
...                 y=13,
...                 geom="<some_geom>"
...             )
...         )
...         session.commit()
...
>>> add_etrango_bus() # doctest: +SKIP
```

Parameters *func* (*func*) – Function to wrap

Notes

Background: using *next_etrango_id()* may cause trouble if tasks are executed simultaneously, cf. <https://github.com/openego/eGon-data/issues/514>

Important: your function requires a way to escape the violation as the loop will not terminate until the error is resolved! In case of eTraGo tables you can use `next_etrango_id()`, see example above.

credentials()

Return local database connection parameters.

Returns *dict* – Complete DB connection information

engine()

Engine for local database.

engine_for

execute_sql (*sql_string*)

Execute a SQL expression given as string.

The SQL expression passed as plain string is convert to a *sqlalchemy.sql.expression.TextClause*.

Parameters *sql_string* (*str*) – SQL expression

execute_sql_script (*script*, *encoding*='utf-8-sig')

Execute a SQL script given as a file name.

Parameters

- **script** (*str*) – Path of the SQL-script
- **encoding** (*str*) – Encoding which is used for the SQL file. The default is “utf-8-sig”.

Returns *None*.

next_etrango_id (*component*)

Select next id value for components in etrango tables

Parameters *component* (*str*) – Name of component

Returns *next_id* (*int*) – Next index value

Notes

To catch concurrent DB commits, consider to use `check_db_unique_violation()` instead.

select_dataframe (*sql*, *index_col*=None, *warning*=True)

Select data from local database as pandas.DataFrame

Parameters

- **sql** (*str*) – SQL query to be executed.
- **index_col** (*str*, *optional*) – Column(s) to set as index(MultiIndex). The default is None.

Returns *df* (*pandas.DataFrame*) – Data returned from SQL statement.

select_geodataframe (*sql*, *index_col*=None, *geom_col*='geom', *epsg*=3035)

Select data from local database as geopandas.GeoDataFrame

Parameters

- **sql** (*str*) – SQL query to be executed.
- **index_col** (*str*, *optional*) – Column(s) to set as index(MultiIndex). The default is None.
- **geom_col** (*str*, *optional*) – column name to convert to shapely geometries. The default is 'geom'.
- **epsg** (*int*, *optional*) – EPSG code specifying output projection. The default is 3035.

Returns `gdf` (*pandas.DataFrame*) – Data returned from SQL statement.

session_scope ()

Provide a transactional scope around a series of operations.

session_scoped (*function*)

Provide a session scope to a function.

Can be used as a decorator like this:

```
>>> @session_scoped
... def get_bind(session):
...     return session.get_bind()
...
>>> get_bind()
Engine(postgresql+psycopg2://egon:***@127.0.0.1:59734/egon-data)
```

Note that the decorated function needs to accept a parameter named *session*, but is called without supplying a value for that parameter because the parameter's value will be filled in by *session_scoped*. Using this decorator allows saving an indentation level when defining such functions but it also has other usages.

submit_comment (*json, schema, table*)

Add comment to table.

We use [Open Energy Metadata](#) standard for describing our data. Metadata is stored as JSON in the table comment.

Parameters

- **json** (*str*) – JSON string reflecting comment
- **schema** (*str*) – The target table's database schema
- **table** (*str*) – Database table on which to put the given comment

10.7 metadata

context ()

Project context information for metadata

Returns *dict* – OEP metadata conform data license information

generate_resource_fields_from_db_table (*schema, table, geom_columns=None*)

Generate a template for the resource fields for metadata from a database table.

For details on the fields see field 14.6.1 of [Open Energy Metadata](#) standard. The fields *name* and *type* are automatically filled, the *description* and *unit* must be filled manually.

Examples

```
>>> from egon.data.metadata import generate_resource_fields_from_db_table
>>> resources = generate_resource_fields_from_db_table(
...     'openstreetmap', 'osm_point', ['geom', 'geom_centroid']
... ) # doctest: +SKIP
```

Parameters

- **schema** (*str*) – The target table's database schema

- **table** (*str*) – Database table on which to put the given comment
- **geom_columns** (*list of str*) – Names of all geometry columns in the table. This is required to return Geometry data type for those columns as SQL Alchemy does not recognize them correctly. Defaults to ['geom'].

Returns *list of dict* – Resource fields

generate_resource_fields_from_sqla_model (*model*)

Generate a template for the resource fields for metadata from a SQL Alchemy model.

For details on the fields see field 14.6.1 of [Open Energy Metadata](#) standard. The fields *name* and *type* are automatically filled, the *description* and *unit* must be filled manually.

Examples

```
>>> from egon.data.metadata import generate_resource_fields_from_sqla_model
>>> from egon.data.datasets.zensus_vg250 import Vg250Sta
>>> resources = generate_resource_fields_from_sqla_model(Vg250Sta)
```

Parameters **model** (*sqlalchemy.ext.declarative.declarative_base()*) – SQLA model

Returns *list of dict* – Resource fields

license_ccby (*attribution*)

License information for Creative Commons Attribution 4.0 International (CC-BY-4.0)

Parameters **attribution** (*str*) – Attribution for the dataset incl. © symbol, e.g. '© GeoBasis-DE / BKG'

Returns *dict* – OEP metadata conform data license information

license_geonutzv (*attribution*)

License information for GeoNutzV

Parameters **attribution** (*str*) – Attribution for the dataset incl. © symbol, e.g. '© GeoBasis-DE / BKG'

Returns *dict* – OEP metadata conform data license information

license_odbl (*attribution*)

License information for Open Data Commons Open Database License (ODbL-1.0)

Parameters **attribution** (*str*) – Attribution for the dataset incl. © symbol, e.g. '© OpenStreetMap contributors'

Returns *dict* – OEP metadata conform data license information

licenses_datenlizenz_deutschland (*attribution*)

License information for Datenlizenz Deutschland

Parameters **attribution** (*str*) – Attribution for the dataset incl. © symbol, e.g. '© GeoBasis-DE / BKG'

Returns *dict* – OEP metadata conform data license information

meta_metadata ()

Meta data on metadata

Returns *dict* – OEP metadata conform metadata on metadata

10.8 subprocess

Extensions to Python's `subprocess` module.

More specifically, this module provides a customized version of `subprocess.run()`, which always sets `check=True`, `capture_output=True`, enhances the raised exceptions string representation with additional output information and makes it slightly more readable when encountered in a stack trace.

exception CalledProcessError (*returncode, cmd, output=None, stderr=None*)

Bases: `subprocess.CalledProcessError`

A more verbose version of `subprocess.CalledProcessError`.

Replaces the standard string representation of a `subprocess.CalledProcessError` with one that has more output and error information and is formatted to be more readable in a stack trace.

run (**args, **kwargs*)

A “safer” version of `subprocess.run()`.

“Safer” in this context means that this version always raises `CalledProcessError` if the process in question returns a non-zero exit status. This is done by setting `check=True` and `capture_output=True`, so you don't have to specify these yourself anymore. You can though, if you want to override these defaults. Other than that, the function accepts the same parameters as `subprocess.run()`.

CHAPTER 11

Indices and tables

- `genindex`
- `modindex`
- `search`

Bibliography

- [NEP2021] Principles for the Expansion Planning of the German Transmission Network <https://www.netzentwicklungsplan.de/>
- [BAST] Bundesanstalt für Straßenwesen, Automatische Zählstellen 2020 (2020). URL https://www.bast.de/DE/Verkehrstechnik/Fachthemen/v2-verkehrszaehlung/Daten/2020_1/Jawe2020.cs
- [Brakelmann2004] H. Brakelmann, Netzverstärkungs-Trassen zur Übertragung von Windenergie: Freileitung oder Kabel? (2004). URL http://www.ets.uni-duisburg-essen.de/download/public/Freileitung_Kabel.pdf
- [Buettner2022] C. Büttner, J. Amme, J. Endres, A. Malla, B. Schachler, I. Cußmann, Open modeling of electricity and heat demand curves for all residential buildings in Germany, Energy Informatics 5 (1) (2022) 21. doi:10.1186/s42162-022-00201-y. URL <https://doi.org/10.1186/s42162-022-00201-y>
- [Census] S. B. (Destatis), Datensatzbeschreibung "Haushalte im 100 Meter-Gitter" (2018). URL https://www.zensus2011.de/SharedDocs/Downloads/DE/Pressemitteilung/DemografischeGrunddaten/Datensatzbeschreibung_Haushalt_100m_Gitter.html
- [DAE_store] Danish Energy Agency, Technology Data – Energy storage, First published 2018 by the Danish Energy Agency and Energinet, URL <https://ens.dk/en/our-services/projections-and-models/technology-data/technology-data-energy-storage>
- [demandregio] F. Gotzens, B. Gillesen, S. Burges, W. Hennings, J. Müller-Kirchenbauer, S. Seim, P. Verwiebe, S. Tobias, F. Jetter, T. Limmer, DemandRegio - Harmonisierung und Entwicklung von Verfahren zur regionalen und zeitlichen Auflösung von Energienachfragen (2020). URL <https://openaccess.ffe.de/10.34805/ffe-119-20>
- [Energierferenzprognose] Prognos AG, Energiewirtschaftliches Institut an der Universität zu Köln, Gesellschaft für Wirtschaftliche Strukturforshung mbH: Entwicklung der Energiemärkte – Energierferenzprognose (2014)
- [EsterlDentzien] Katharina Esterl, Hannah Dentzien, Integration von Demand Side Management in eTraGo, Student Work, Hochschule Flensburg, URL https://ego-n.org/theses/2021_SP_Esterl_Dentzien_DSM-eTraGo.pdf
- [eXtremOS] A. Guminski, C. Fiedler, S. Kigle, C. Pellingner, P. Dossow, K. Ganz, F. Jetter, T. Kern, T. Limmer, A. Murmann, J. Reinhard, T. Schmid, T. Schmidt-Achert, S. von Roon, eXtremOS Summary Report (2021). doi:<https://doi.org/10.34805/ffe-24-21>.
- [FfE2017] Flexibilisierung der Kraft-Wärme-Kopplung; 2017; Forschungsstelle für Energiewirtschaft e.V. (FfE)
- [Heitkoetter] Wilko Heitkoetter, Bruno U. Schyska, Danielle Schmidt, Wided Medjroubi, Thomas Vogt, Carsten Agert, Assessment of the regionalised demand response potential in Germany using an open source tool

- and dataset, *Advances in Applied Energy* (2021), URL <https://www.sciencedirect.com/science/article/pii/S2666792420300019>
- [Helfenbein2021] K. Helfenbein, Analyse des Einflusses netzdienlicher Ladestrategien auf Verteilnetze aufgrund der zunehmenden Netzintegration von Elektrofahrzeugen, Master's thesis, Hochschule für Technik und Wirtschaft Berlin, URL <https://reiner-lemoine-institut.de/analyse-einflussesnetzdienlicher-ladestrategien-verteilnetze-zunehmender-netzintegration-elektrofahrzeuge-helfenbein>
- [Hotmaps] S. Pezzutto, S. Zambotti, S. Croce, P. Zambelli, G. Garegnani, C. Scaramuzzino, R. P. Pascuas, A. Zubaryeva, F. Haas, D. Exner, A. Mueller, M. Hartner, T. Fleiter, A.-L. Klingler, M. Kuehnbach, P. Manz, S. Marwitz, M. Rehfeldt, J. Steinbach, E. Popovski, Hotmaps project, d2.3 wp2 report – open data set for the eu28 (2018). URL www.hotmaps-project.eu
- [Huelk2017] L. Hülk, L. Wienholt, I. Cußmann, U.P. Müller, C. Matke, E. Kötter, Allocation of annual electricity consumption and power generation capacities across multiple voltage levels in a high spatial resolution, *International Journal of Sustainable Energy Planning and Management* Vol. 13 2017 79–92. URL <https://journals.aau.dk/index.php/sepm/article/view/1833>
- [Kleinhans] D. Kleinhans, Towards a systematic characterization of the potential of demand side management, arXiv (2014), doi: 10.48550/ARXIV.1401.4121. URL <https://arxiv.org/abs/1401.4121>
- [MaStR] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen, Marktstammdatenregister - Datendownload (Nov. 2022). URL <https://www.marktstammdatenregister.de/MaStR/Datendownload>
- [MiD2017] Bundesministerium für Digitales und Verkehr, Mobilität in Deutschland 2017 (2017). URL <https://daten.clearingstelle-verkehr.de/279/>
- [Mueller2018] U. Mueller, L. Wienholt, D. Kleinhans, I. Cusmann, W.-D. Bunke, G. Pleßmann, J. Wendiggensen 2018 J. Phys.: Conf. Ser. 977 012003, DOI 10.1088/1742-6596/977/1/012003
- [NEP2021] Übertragungsnetzbetreiber Deutschland (2021): *Netzentwicklungsplan Strom 2035*, Version 2021, 1. Entwurf. 2021.
- [NOW2020] Nationale Leitstelle Ladeinfrastruktur, Ladeinfrastruktur nach 2025/2030: Szenarien für den Markthochlauf (2020). URL https://www.now-gmbh.de/wp-content/uploads/2020/11/Studie_Ladeinfrastruktur-nach-2025-2.pdf
- [OSM] Geofabrik GmbH and OpenStreetMap-Contributors, OpenStreetMap Data Extracts, Stand 01.01.2022 (2022). URL <https://download.geofabrik.de/europe/germany-220101.osm.pbf>
- [Peta] Europa-Universität Flensburg, Halmstad University and Aalborg University, Pan-European Thermal Atlas - Residential heat demand (2021). URL <https://s-eenergies-open-data-euf.hub.arcgis.com/maps/d7d18b63250240a49eb81db972aa573e/about>
- [RegioStaR7_2020] Bundesministerium für Digitales und Verkehr, Regionalstatistische Raumtypologie (RegioStaR7), Gebietsstand 2020 (2020). URL <https://mcloud.de/web/guest/suche/-/results/detail/536149D1-2902-4975-9F7D-253191C0AD07>
- [Schmidt2018] D. Schmidt, Supplementary material to the masters thesis: NUTS-3 Regionalization of Industrial Load Shifting Potential in Germany using a Time-Resolved Model (Nov. 2019). doi:10.5281/zenodo.3613767. URL <https://doi.org/10.5281/zenodo.3613767>
- [sEEnergies] T. Fleiter, P. Manz, N. Neuwirth, F. Mildner, K. Persson, U. AND Kermeli, W. Crijns-Graus, C. Rutten, seenergies d5.1 dataset web-app.seenergies arcgis online web-apps hosted by europa-universität flensburg (2020). URL <https://tinyurl.com/sEEnergies-D5-1>
- [TYNDP] European Network of Transmission System Operators for Electricity, European Network of Transmission System Operators for Gas, Ten-Year Network Development Plans - “TYNDP 2020 Scenarios” (2020)

a

`egon.data.airflow`, 63

c

`egon.data.cli`, 63

`egon.data.config`, 63

`egon.data.datasets.calculate_dlr`, 68

`egon.data.datasets.ch4_prod`, 69

`egon.data.datasets.ch4_storages`, 70

`egon.data.datasets.chp`, 116

`egon.data.datasets.chp.match_nep`, 113

`egon.data.datasets.chp.small_chp`, 114

`egon.data.datasets.chp_etrago`, 72

d

`egon.data.datasets`, 213

`egon.data.datasets.data_bundle`, 119

`egon.data.datasets.database`, 72

`egon.data.datasets.demandregio`, 119

`egon.data.datasets.demandregio.install_dsaaggregator`, 119

`egon.data.datasets.district_heating_areas`, 122

`egon.data.datasets.district_heating_areas.plot`, 122

`egon.data.datasets.DSM_cts_ind`, 64

`egon.data.db`, 214

e

`egon.data`, 63

`egon.data.datasets.electrical_neighbours`, 72

`egon.data.datasets.electricity_demand`, 126

`egon.data.datasets.electricity_demand.temporal`, 126

`egon.data.datasets.electricity_demand_etrago`, 75

`egon.data.datasets.electricity_demand_timeseries`, 146

`egon.data.datasets.electricity_demand_timeseries.ct`, 127

`egon.data.datasets.electricity_demand_timeseries.h`, 132

`egon.data.datasets.electricity_demand_timeseries.h`, 136

`egon.data.datasets.electricity_demand_timeseries.ma`, 145

`egon.data.datasets.electricity_demand_timeseries.t`, 145

`egon.data.datasets.era5`, 76

`egon.data.datasets.etrago_helpers`, 77

`egon.data.datasets.etrago_setup`, 78

f

`egon.data.datasets.fill_etrago_gen`, 86

`egon.data.datasets.fix_ehv_subnetworks`, 87

g

`egon.data.datasets.gas_aggregator`,

`egon.data.datasets.gas_areas`, 87

`egon.data.datasets.gas_grid`, 89

`egon.data.datasets.gas_neighbours`, 151

`egon.data.datasets.gas_neighbours.eGon100RE`, 147

`egon.data.datasets.gas_neighbours.eGon2035`, 148

`egon.data.datasets.gas_neighbours.gas_abroad`, 151

`egon.data.datasets.generate_voronoi`, 92

h

`egon.data.datasets.heat_demand`, 152

`egon.data.datasets.heat_demand_europe`,

`temporal`, 92

`egon.data.datasets.heat_demand_timeseries`, 157

`egon.data.datasets.heat_demand_timeseries.daily`, 154

egon.data.datasets.heat_demand_timeseries, 155
egon.data.datasets.heat_demand_timeseries.service_sector, 156
egon.data.datasets.heat_etrago, 160
egon.data.datasets.heat_etrago.hts_etrago, 159
egon.data.datasets.heat_etrago.power_to_heat, 159
egon.data.datasets.heat_supply, 171
egon.data.datasets.heat_supply.district_heating, 161
egon.data.datasets.heat_supply.geothermal, 163
egon.data.datasets.heat_supply.individual_heating, 163
egon.data.datasets.hydrogen_etrago, 176
egon.data.datasets.hydrogen_etrago.bus, 173
egon.data.datasets.hydrogen_etrago.h2_grid, 174
egon.data.datasets.hydrogen_etrago.h2_to_ch4, 174
egon.data.datasets.hydrogen_etrago.power_to_h2, 175
egon.data.datasets.hydrogen_etrago.storage, 176

i
egon.data.datasets.industrial_gas_demand, 93
egon.data.datasets.industrial_sites, 179
egon.data.datasets.industry, 182
egon.data.datasets.industry.temporal, 181

l
egon.data.datasets.loadarea, 184
egon.data.datasets.low_flex_scenario, 185

m
egon.data.datasets.mastr, 96
egon.data.datasets.mv_grid_districts, 97
egon.data.metadata, 217

o
egon.data.datasets.osm, 185
egon.data.datasets.osm_buildings_streets, 186
egon.data.datasets.osmtgmod, 188
egon.data.datasets.osmtgmod.substation, 187

p
egon.data.datasets.p_idp_pool, 189
egon.data.datasets.power_etrago, 188
egon.data.datasets.power_etrago.match_ocgt, 188
egon.data.datasets.power_plants, 202
egon.data.datasets.power_plants.assign_weather_data, 189
egon.data.datasets.power_plants.conventional, 189
egon.data.datasets.power_plants.mastr, 190
egon.data.datasets.power_plants.pv_ground_mounted, 191
egon.data.datasets.power_plants.pv_rooftop, 191
egon.data.datasets.power_plants.pv_rooftop_building, 191
egon.data.datasets.power_plants.wind_farms, 201
egon.data.datasets.power_plants.wind_offshore, 202
egon.data.datasets.pypsaeursec, 204

r
egon.data.datasets.re_potential_areas, 204
egon.data.datasets.renewable_feedin, 101

s
egon.data.datasets.saltcavern, 205
egon.data.datasets.sanity_checks, 103
egon.data.datasets.scenario_capacities, 106
egon.data.datasets.scenario_parameters, 206
egon.data.datasets.scenario_parameters.parameters, 205
egon.data.datasets.society_prognosis, 108
egon.data.datasets.storages, 209
egon.data.datasets.storages.home_batteries, 207
egon.data.datasets.storages.pumped_hydro, 208
egon.data.datasets.storages_etrago, 210
egon.data.datasets.substation, 210
egon.data.datasets.substation_voronoi, 109
egon.data.subprocess, 219

t
egon.data.datasets.tyndp, 109

V

`egon.data.datasets.vg250`, [211](#)
`egon.data.datasets.vg250_mv_grid_districts`,
[110](#)

Z

`egon.data.datasets.zensus`, [212](#)
`egon.data.datasets.zensus_mv_grid_districts`,
[110](#)
`egon.data.datasets.zensus_vg250`, [110](#)

A

- [a2035_capacity](#) (*NEP2021ConvPowerPlants* attribute), 106
[a2035_chp](#) (*NEP2021ConvPowerPlants* attribute), 106
[adapt_numpy_float64\(\)](#) (in module *egon.data.datasets.heat_supply.individual_heating*), 166
[adapt_numpy_int64\(\)](#) (in module *egon.data.datasets.heat_supply.individual_heating*), 166
[add_ags_to_buildings\(\)](#) (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 193
[add_ags_to_gens\(\)](#) (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 193
[add_buildings_meta_data\(\)](#) (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 193
[add_bus\(\)](#) (in module *egon.data.datasets.fix_ehv_subnetworks*), 87
[add_bus_ids_sq\(\)](#) (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 194
[add_commissioning_date\(\)](#) (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 194
[add_line\(\)](#) (in module *egon.data.datasets.fix_ehv_subnetworks*), 87
[add_marginal_costs\(\)](#) (in module *egon.data.datasets.fill_etrago_gen*), 86
[add_metadata\(\)](#) (in module *egon.data.datasets.district_heating_areas*), 123
[add_metadata\(\)](#) (in module *egon.data.datasets.heat_demand*), 152
[add_metadata\(\)](#) (in module *egon.data.datasets.osm*), 185
[add_metadata\(\)](#) (in module *egon.data.datasets.osm_buildings_streets*), 186
[add_metadata\(\)](#) (in module *egon.data.datasets.vg250*), 212
[add_metadata_vg250_gem_pop\(\)](#) (in module *egon.data.datasets.zensus_vg250*), 113
[add_metadata_zensus_inside_ger\(\)](#) (in module *egon.data.datasets.zensus_vg250*), 113
[add_overlay_id_to_buildings\(\)](#) (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 194
[add_trafo\(\)](#) (in module *egon.data.datasets.fix_ehv_subnetworks*), 87
[add_weather_cell_id\(\)](#) (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 194
[address](#) (*HotmapsIndustrialSites* attribute), 179
[address](#) (*IndustrialSites* attribute), 179
[address](#) (*SeenergiesIndustrialSites* attribute), 180
[ade](#) (*Vg250Gem* attribute), 111
[ade](#) (*Vg250Lan* attribute), 192
[ade](#) (*Vg250Sta* attribute), 112
[adjust_cts_ind_nep\(\)](#) (in module *egon.data.datasets.demandregio*), 120
[adjust_ind_pes\(\)](#) (in module *egon.data.datasets.demandregio*), 120
[adjust_renew_feedin_table\(\)](#) (in module *egon.data.datasets.fill_etrago_gen*), 86
[adjust_residential_heat_to_zensus\(\)](#) (in module *egon.data.datasets.heat_demand*), 152
[adjust_to_demand_regio_nuts3_annual\(\)](#) (in module *egon.data.datasets.electricity_demand_timeseries.hh*), 139
[adjust_zensus_misc\(\)](#) (in module *egon.data.datasets.zensus*), 212
[aggr_nep_capacities\(\)](#) (in module *egon.data.datasets.scenario_capacities*),

107
 aggregate_components() (in module *egon.data.datasets.DSM_cts_ind*), 65
 aggregate_residential_and_cts_profiles() (in module *egon.data.datasets.heat_supply.individual_heating*), 166
 ags (*Vg250Gem* attribute), 111
 ags (*Vg250Lan* attribute), 192
 ags (*Vg250Sta* attribute), 112
 ags_0 (*HvmvSubstPerMunicipality* attribute), 97
 ags_0 (*Vg250Gem* attribute), 111
 ags_0 (*Vg250GemClean* attribute), 98
 ags_0 (*Vg250GemPopulation* attribute), 112
 ags_0 (*Vg250Lan* attribute), 192
 ags_0 (*Vg250Sta* attribute), 112
 ags_0 (*VoronoiMunicipalityCuts* attribute), 99
 ags_0 (*VoronoiMunicipalityCutsAssigned* attribute), 99
 ags_0 (*VoronoiMunicipalityCutsBase* attribute), 99
 allocate_conventional_non_chp_power_plants() (in module *egon.data.datasets.power_plants*), 203
 allocate_home_batteries_to_buildings() (in module *egon.data.datasets.storages.home_batteries*), 208
 allocate_other_power_plants() (in module *egon.data.datasets.power_plants*), 203
 allocate_pumped_hydro_eGon100RE() (in module *egon.data.datasets.storages*), 209
 allocate_pumped_hydro_eGon2035() (in module *egon.data.datasets.storages*), 209
 allocate_pv() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 194
 allocate_pv_home_batteries_to_grids() (in module *egon.data.datasets.storages*), 210
 allocate_scenarios() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 195
 allocate_to_buildings() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 195
 amenities_without_buildings() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 130
 amenity (*OsmBuildingsFiltered* attribute), 192
 annual_demand_generator() (in module *egon.data.datasets.heat_demand_timeseries.idp_pool*), 155
 annual_tonnes (*SchmidtIndustrialSites* attribute), 180
 annualize_capital_costs() (in module *egon.data.datasets.scenario_parameters.parameters*), 205
 application (*EgonDemandregioSitesIndElectricityDsmTimeseries* attribute), 64
 application (*SchmidtIndustrialSites* attribute), 180
 apply_voltage_level_thresholds() (in module *egon.data.datasets.storages.pumped_hydro*), 208
 area (*MvGridDistricts* attribute), 98
 area (*MvGridDistrictsDissolved* attribute), 98
 area (*OsmBuildingsFiltered* attribute), 192
 area (*OsmBuildingsSynthetic* attribute), 134
 area_grouping() (in module *egon.data.datasets.district_heating_areas*), 123
 area_ha (*HvmvSubstPerMunicipality* attribute), 97
 area_ha (*OsmPolygonUrban* attribute), 184
 area_ha (*Vg250GemClean* attribute), 98
 area_ha (*Vg250GemPopulation* attribute), 112
 area_id (*EgonDistrictHeatingAreas* attribute), 123
 area_id (*EgonTimeseriesDistrictHeating* attribute), 123
 area_id (*MapZensusDistrictHeatingAreas* attribute), 123
 area_km2 (*Vg250GemPopulation* attribute), 112
 ars (*Vg250Gem* attribute), 111
 ars (*Vg250Lan* attribute), 192
 ars (*Vg250Sta* attribute), 112
 ars_0 (*Vg250Gem* attribute), 111
 ars_0 (*Vg250Lan* attribute), 192
 ars_0 (*Vg250Sta* attribute), 112
 assign_bus_id() (in module *egon.data.datasets.power_plants*), 203
 assign_electrical_bus() (in module *egon.data.datasets.heat_etrago.power_to_heat*), 159
 assign_gas_bus_id() (in module *egon.data.db*), 214
 assign_heat_bus() (in module *egon.data.datasets.chp*), 117
 assign_hh_demand_profiles_to_cells() (in module *egon.data.datasets.electricity_demand_timeseries.hh_profiles*), 139
 assign_substation_municipality_fragments() (in module *egon.data.datasets.mv_grid_districts*), 113
 assign_use_case() (in module *egon.data.datasets.chp.small_chp*), 114
 assign_voltage_level() (in module *egon.data.datasets.heat_etrago.power_to_heat*), 159
 assign_voltage_level() (in module *egon.data.datasets.power_plants*), 203
 assign_voltage_level_by_capacity() (in module *egon.data.datasets.power_plants*), 203
 assign_voltage_level_to_buildings() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 130

130

B

- b (*EgonPfHvLine* attribute), 80
- b (*EgonPfHvTransformer* attribute), 84
- b2035_capacity (*NEP2021ConvPowerPlants* attribute), 106
- b2035_chp (*NEP2021ConvPowerPlants* attribute), 106
- b2040_capacity (*NEP2021ConvPowerPlants* attribute), 106
- b2040_chp (*NEP2021ConvPowerPlants* attribute), 107
- backup_gas_boilers() (in module *egon.data.datasets.heat_supply.district_heating*), 161
- backup_resistive_heaters() (in module *egon.data.datasets.heat_supply.district_heating*), 161
- bem (*HvmvSubstPerMunicipality* attribute), 97
- bem (*Vg250Gem* attribute), 111
- bem (*Vg250GemClean* attribute), 98
- bem (*Vg250GemPopulation* attribute), 112
- bem (*Vg250Lan* attribute), 192
- bem (*Vg250Sta* attribute), 112
- bez (*HvmvSubstPerMunicipality* attribute), 97
- bez (*Vg250Gem* attribute), 111
- bez (*Vg250GemClean* attribute), 98
- bez (*Vg250GemPopulation* attribute), 112
- bez (*Vg250Lan* attribute), 192
- bez (*Vg250Sta* attribute), 112
- bnetza_id (*NEP2021ConvPowerPlants* attribute), 107
- bsg (*Vg250Gem* attribute), 111
- bsg (*Vg250Lan* attribute), 193
- bsg (*Vg250Sta* attribute), 112
- build_year (*EgonPfHvGenerator* attribute), 79
- build_year (*EgonPfHvLine* attribute), 80
- build_year (*EgonPfHvLink* attribute), 81
- build_year (*EgonPfHvStorage* attribute), 82
- build_year (*EgonPfHvStore* attribute), 83
- build_year (*EgonPfHvTransformer* attribute), 84
- building (*OsmBuildingsFiltered* attribute), 192
- building (*OsmBuildingsSynthetic* attribute), 134
- building_area_range_per_cap_range() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 195
- building_id (*BuildingElectricityPeakLoads* attribute), 134
- building_id (*BuildingHeatPeakLoads* attribute), 129, 166
- building_id (*EgonCtsElectricityDemandBuildingShare* attribute), 130
- building_id (*EgonCtsHeatDemandBuildingShare* attribute), 130
- building_id (*EgonHeatTimeseries* attribute), 155
- building_id (*EgonHomeBatteries* attribute), 208
- building_id (*EgonHpCapacityBuildings* attribute), 166
- building_id (*EgonIndividualHeatingPeakLoads* attribute), 157
- building_id (*EgonMapZensusMygdBuildings* attribute), 145
- building_id (*EgonPowerPlantPvRoofBuilding* attribute), 192
- building_id (*HouseholdElectricityProfilesOfBuildings* attribute), 134
- BuildingElectricityPeakLoads (class in *egon.data.datasets.electricity_demand_timeseries.hh_buildings*), 134
- BuildingHeatPeakLoads (class in *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 129
- BuildingHeatPeakLoads (class in *egon.data.datasets.heat_supply.individual_heating*), 165
- buildings_with_amenities() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 130
- buildings_without_amenities() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 130
- bus (*DemandCurvesOsmIndustry* attribute), 182
- bus (*DemandCurvesSitesIndustry* attribute), 182
- bus (*EgonDemandregioSitesIndElectricityDsmTimeseries* attribute), 64
- bus (*EgonEtragoElectricityCtsDsmTimeseries* attribute), 65
- bus (*EgonOsmIndLoadCurvesIndividualDsmTimeseries* attribute), 65
- bus (*EgonPfHvGenerator* attribute), 79
- bus (*EgonPfHvLoad* attribute), 82
- bus (*EgonPfHvStorage* attribute), 82
- bus (*EgonPfHvStore* attribute), 83
- bus (*EgonSitesIndLoadCurvesIndividualDsmTimeseries* attribute), 65
- bus0 (*EgonPfHvBusmap* attribute), 78
- bus0 (*EgonPfHvLine* attribute), 80
- bus0 (*EgonPfHvLink* attribute), 81
- bus0 (*EgonPfHvTransformer* attribute), 84
- bus1 (*EgonPfHvBusmap* attribute), 78
- bus1 (*EgonPfHvLine* attribute), 80
- bus1 (*EgonPfHvLink* attribute), 81
- bus1 (*EgonPfHvTransformer* attribute), 84
- bus_id (*DemandCurvesOsmIndustryIndividual* attribute), 182
- bus_id (*DemandCurvesSitesIndustryIndividual* attribute), 183
- bus_id (*EgonCtsElectricityDemandBuildingShare* attribute), 130
- bus_id (*EgonCtsHeatDemandBuildingShare* attribute), 130

- 130
 bus_id (*EgonEhvSubstation* attribute), 187
 bus_id (*EgonEhvSubstationVoronoi* attribute), 109
 bus_id (*EgonEhvTransferBuses* attribute), 210
 bus_id (*EgonEtragoElectricityCts* attribute), 126
 bus_id (*EgonEtragoElectricityHouseholds* attribute), 138
 bus_id (*EgonEtragoHeatCts* attribute), 157
 bus_id (*EgonEtragoTimeseriesIndividualHeating* attribute), 157, 166
 bus_id (*EgonHomeBatteries* attribute), 208
 bus_id (*EgonHvmvSubstation* attribute), 187
 bus_id (*EgonHvmvSubstationVoronoi* attribute), 109
 bus_id (*EgonHvmvTransferBuses* attribute), 211
 bus_id (*EgonMapZensusMvgdBuildings* attribute), 145
 bus_id (*EgonPfHvBus* attribute), 78
 bus_id (*EgonPfHvBusTimeseries* attribute), 78
 bus_id (*EgonPfHvGasVoronoi* attribute), 87
 bus_id (*EgonPowerPlantPvRoofBuilding* attribute), 192
 bus_id (*EgonPowerPlants* attribute), 202
 bus_id (*EgonStorages* attribute), 209
 bus_id (*MapMvgriddistrictsVg250* attribute), 110
 bus_id (*MapZensusGridDistricts* attribute), 110
 bus_id (*MvGridDistricts* attribute), 98
 bus_id (*MvGridDistrictsDissolved* attribute), 98
 bus_id (*VoronoiMunicipalityCuts* attribute), 99
 bus_id (*VoronoiMunicipalityCutsAssigned* attribute), 99
 bus_id (*VoronoiMunicipalityCutsBase* attribute), 99
 buses() (in module *egon.data.datasets.electrical_neighbours*), 73
 buses() (in module *egon.data.datasets.heat_etrago*), 161
- ## C
- c2035_capacity (*NEP2021ConvPowerPlants* attribute), 107
 c2035_chp (*NEP2021ConvPowerPlants* attribute), 107
 cables (*EgonPfHvLine* attribute), 80
 calc_building_demand_profile_share() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 130
 calc_capacities() (in module *egon.data.datasets.electrical_neighbours*), 73
 calc_capacities() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 148
 calc_capacity_per_year() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 148
 calc_census_cell_share() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 131
 calc_ch4_storage_capacities() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 149
 calc_cts_building_profiles() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 131
 calc_geothermal_costs() (in module *egon.data.datasets.heat_supply.geothermal*), 163
 calc_geothermal_potentials() (in module *egon.data.datasets.heat_supply.geothermal*), 163
 calc_global_ch4_demand() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 149
 calc_global_power_to_h2_demand() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 149
 calc_ind_site_timeseries() (in module *egon.data.datasets.DSM_cts_ind*), 66
 calc_load_curve() (in module *egon.data.datasets.electricity_demand.temporal*), 126
 calc_load_curves_cts() (in module *egon.data.datasets.electricity_demand.temporal*), 126
 calc_load_curves_ind_osm() (in module *egon.data.datasets.industry.temporal*), 181
 calc_load_curves_ind_sites() (in module *egon.data.datasets.industry.temporal*), 181
 calc_residential_heat_profiles_per_mvgd() (in module *egon.data.datasets.heat_supply.individual_heating*), 166
 calc_usable_geothermal_potential() (in module *egon.data.datasets.heat_supply.geothermal*), 163
 calculate_and_map_salt cavern_storage_potential() (in module *egon.data.datasets.hydrogen_etrago.storage*), 176
 calculate_building_load_factor() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 196
 calculate_ch4_grid_capacities() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 149
 calculate_crossbordering_gas_grid_capacities_eGon100RE() (in module *egon.data.datasets.gas_neighbours.eGon100RE*), 147
 Calculate_dlr (class in *egon.data.datasets.calculate_dlr*), 68
 calculate_max_pv_cap_per_building() (in

module egon.data.datasets.power_plants.pv_rooftop_buildings (*EgonStorages* attribute), 209
 196
calculate_ocgt_capacities() (in *module egon.data.datasets.gas_neighbours.eGon2035*), 149
calculate_potentials() (in *module egon.data.datasets.DSM_cts_ind*), 66
CalledProcessError, 219
calulate_peak_load() (in *module egon.data.datasets.heat_demand_timeseries*), 158
cap_per_bus_id() (in *module egon.data.datasets.power_plants.pv_rooftop_buildings*), 196
cap_share_per_cap_range() (in *module egon.data.datasets.power_plants.pv_rooftop_buildings*), 196
capacity (*EgonDistrictHeatingSupply* attribute), 171
capacity (*EgonHomeBatteries* attribute), 208
capacity (*EgonIndividualHeatingSupply* attribute), 172
capacity (*EgonPowerPlantPvRoofBuilding* attribute), 192
capacity (*EgonScenarioCapacities* attribute), 106
capacity (*NEP2021ConvPowerPlants* attribute), 107
capacity_per_district_heating_category() (in *module egon.data.datasets.heat_supply.district_heating* attribute), 162
capacity_production (*SchmidtIndustrialSites* attribute), 180
capital_cost (*EgonPfHvGenerator* attribute), 79
capital_cost (*EgonPfHvLine* attribute), 80
capital_cost (*EgonPfHvLink* attribute), 81
capital_cost (*EgonPfHvStorage* attribute), 82
capital_cost (*EgonPfHvStore* attribute), 83
capital_cost (*EgonPfHvTransformer* attribute), 84
carrier (*EgonChp* attribute), 117
carrier (*EgonDistrictHeatingSupply* attribute), 171
carrier (*EgonEtragoTimeseriesIndividualHeating* attribute), 166
carrier (*EgonIndividualHeatingSupply* attribute), 172
carrier (*EgonMaStRConventinalWithoutChp* attribute), 117
carrier (*EgonPfHvBus* attribute), 78
carrier (*EgonPfHvGasVoronoi* attribute), 87
carrier (*EgonPfHvGenerator* attribute), 79
carrier (*EgonPfHvLine* attribute), 80
carrier (*EgonPfHvLink* attribute), 81
carrier (*EgonPfHvLoad* attribute), 82
carrier (*EgonPfHvStorage* attribute), 82
carrier (*EgonPfHvStore* attribute), 83
carrier (*EgonPowerPlants* attribute), 202
carrier (*EgonRenewableFeedIn* attribute), 76
carrier (*EgonScenarioCapacities* attribute), 106
carrier (*NEP2021ConvPowerPlants* attribute), 107
carrier_nep (*NEP2021ConvPowerPlants* attribute), 107
cascade_heat_supply() (in *module egon.data.datasets.heat_supply.district_heating*), 162
cascade_heat_supply_indiv() (in *module egon.data.datasets.heat_supply.individual_heating*), 167
cascade_per_technology() (in *module egon.data.datasets.heat_supply.district_heating*), 162
cascade_per_technology() (in *module egon.data.datasets.heat_supply.individual_heating*), 167
catch_missing_buidings() (in *module egon.data.datasets.heat_supply.individual_heating*), 167
category (*EgonDistrictHeatingSupply* attribute), 171
category (*EgonIndividualHeatingSupply* attribute), 172
cell_count (*Vg250GemPopulation* attribute), 112
cell_id (*EgonDestatisZensusHouseholdPerHaRefined* attribute), 138
cell_id (*HouseholdElectricityProfilesInCensusCells* attribute), 138
cell_id (*HouseholdElectricityProfilesOfBuildings* attribute), 134
cell_id (*OsmBuildingsSynthetic* attribute), 134
cell_profile_ids (*HouseholdElectricityProfilesInCensusCells* attribute), 138
cells_with_cts_demand_only() (in *module egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 131
census_cells_melt() (in *module egon.data.datasets.loadarea*), 184
central_buses_egon100() (in *module egon.data.datasets.electrical_neighbours*), 73
central_transformer() (in *module egon.data.datasets.electrical_neighbours*), 73
ch4_bus_id (*EgonChp* attribute), 117
ch4_nodes_number_G() (in *module egon.data.datasets.gas_grid*), 89
CH4Production (class in *module egon.data.datasets.ch4_prod*), 69
CH4Storages (class in *module egon.data.datasets.ch4_storages*), 70
characteristics_code (*EgonDestatisZensusHouseholdPerHaRefined* attribute), 138
check_carriers() (in *module egon.data.datasets.etrago_setup*), 85

`check_db_unique_violation()` (in module *egon.data.db*), 215
`check_version()` (*Dataset method*), 213
`choose_transformer()` (in module *egon.data.datasets.electrical_neighbours*), 73
`Chp` (class in *egon.data.datasets.chp*), 116
`chp` (*NEP2021ConvPowerPlants attribute*), 107
`ChpEtrago` (class in *egon.data.datasets.chp_etrago*), 72
`city` (*EgonMaStRConventinalWithoutChp attribute*), 117
`city` (*HotmapsIndustrialSites attribute*), 179
`city` (*NEP2021ConvPowerPlants attribute*), 107
`citycode` (*HotmapsIndustrialSites attribute*), 179
`clean()` (in module *egon.data.datasets.electricity_demand_timeseries.hh_profiles*), 139
`clean_database()` (in module *egon.data.datasets.pypsaeursec*), 204
`clean_mastr_data()` (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 196
`cleaning_and_preperation()` (in module *egon.data.datasets.vg250*), 212
`climate_zone` (*EgonDailyHeatDemandPerClimateZone attribute*), 154
`climate_zone` (*EgonMapZensusClimateZones attribute*), 155
`clone_and_install()` (in module *egon.data.datasets.demandregio.install_disaggregator*), 119
`co2_emissions` (*EgonPfHvCarrier attribute*), 78
`color` (*EgonPfHvCarrier attribute*), 79
`commentary` (*EgonPfHvCarrier attribute*), 79
`commissioned` (*NEP2021ConvPowerPlants attribute*), 107
`committable` (*EgonPfHvGenerator attribute*), 79
`companyname` (*HotmapsIndustrialSites attribute*), 179
`companyname` (*IndustrialSites attribute*), 179
`companyname` (*SeenergiesIndustrialSites attribute*), 180
`component` (*EgonScenarioCapacities attribute*), 106
`consistency()` (in module *egon.data.datasets.fill_etrago_gen*), 86
`context()` (in module *egon.data.metadata*), 217
`control` (*EgonPfHvGenerator attribute*), 79
`control` (*EgonPfHvStorage attribute*), 82
`copy_and_modify_buses()` (in module *egon.data.datasets.etrago_helpers*), 77
`copy_and_modify_links()` (in module *egon.data.datasets.etrago_helpers*), 77
`copy_and_modify_stores()` (in module *egon.data.datasets.etrago_helpers*), 77
`count_hole` (*HvmvSubstPerMunicipality attribute*), 97
`count_hole` (*Vg250GemClean attribute*), 98
`country` (*EgonPfHvBus attribute*), 78
`country` (*HotmapsIndustrialSites attribute*), 179
`country` (*SeenergiesIndustrialSites attribute*), 180
`create()` (in module *egon.data.datasets.heat_demand_timeseries.idp_pool*), 155
`create_buildings_filtered_all_zensus_mapping()` (in module *egon.data.datasets.osm_buildings_streets*), 186
`create_buildings_filtered_zensus_mapping()` (in module *egon.data.datasets.osm_buildings_streets*), 186
`create_buildings_residential_zensus_mapping()` (in module *egon.data.datasets.osm_buildings_streets*), 186
`create_buildings_temp_tables()` (in module *egon.data.datasets.osm_buildings_streets*), 186
`create_combined_zensus_table()` (in module *egon.data.datasets.zensus*), 212
`create_district_heating_profile()` (in module *egon.data.datasets.heat_demand_timeseries*), 158
`create_district_heating_profile_python_like()` (in module *egon.data.datasets.heat_demand_timeseries*), 158
`create_dsm_components()` (in module *egon.data.datasets.DSM_cts_ind*), 66
`create_gas_voronoi_table()` (in module *egon.data.datasets.gas_areas*), 88
`create_individual_heat_per_mv_grid()` (in module *egon.data.datasets.heat_demand_timeseries*), 158
`create_individual_heating_peak_loads()` (in module *egon.data.datasets.heat_demand_timeseries*), 158
`create_individual_heating_profile_python_like()` (in module *egon.data.datasets.heat_demand_timeseries*), 158
`create_landuse_table()` (in module *egon.data.datasets.loadarea*), 184
`create_missing_zensus_data()` (in module *egon.data.datasets.electricity_demand_timeseries.hh_profiles*), 140
`create_scenario_table()` (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 197
`create_sql_functions()` (in module *egon.data.datasets.substation*), 211
`create_synthetic_buildings()` (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 131
`create_table()` (in module

egon.data.datasets.DSM_cts_ind), 66
create_table() (in module *egon.data.datasets.electricity_demand.temporal*), 126
create_table() (in module *egon.data.datasets.scenario_capacities*), 107
create_table() (in module *egon.data.datasets.scenario_parameters*), 207
create_table() (in module *egon.data.datasets.storages.home_batteries*), 208
create_tables() (in module *egon.data.datasets.chp*), 118
create_tables() (in module *egon.data.datasets.demandregio*), 120
create_tables() (in module *egon.data.datasets.district_heating_areas*), 123
create_tables() (in module *egon.data.datasets.electricity_demand*), 127
create_tables() (in module *egon.data.datasets.era5*), 76
create_tables() (in module *egon.data.datasets.etrango_setup*), 85
create_tables() (in module *egon.data.datasets.heat_supply*), 172
create_tables() (in module *egon.data.datasets.industrial_sites*), 181
create_tables() (in module *egon.data.datasets.industry*), 183
create_tables() (in module *egon.data.datasets.osmtgmod.substation*), 188
create_tables() (in module *egon.data.datasets.power_plants*), 203
create_tables() (in module *egon.data.datasets.re_potential_areas*), 205
create_tables() (in module *egon.data.datasets.society_prognosis*), 109
create_tables() (in module *egon.data.datasets.storages*), 210
create_tables() (in module *egon.data.datasets.substation*), 211
create_tables() (in module *egon.data.datasets.substation_voronoi*), 109
create_tables() (in module *egon.data.datasets.vg250_mv_grid_districts*), 110
create_timeseries_for_building() (in module *egon.data.datasets.heat_demand_timeseries*), 158
create_voronoi() (in module *egon.data.datasets.gas_areas*), 88
create_zensus_misc_tables() (in module *egon.data.datasets.zensus*), 212
create_zensus_pop_table() (in module *egon.data.datasets.zensus*), 212
credentials() (in module *egon.data.db*), 216
cross_border_lines() (in module *egon.data.datasets.electrical_neighbours*), 74
cts_buildings() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 131
cts_data_import() (in module *egon.data.datasets.DSM_cts_ind*), 66
cts_demand_per_aggregation_level() (in module *egon.data.datasets.heat_demand_timeseries.service_sector*), 156
CTS_demand_scale() (in module *egon.data.datasets.heat_demand_timeseries.service_sector*), 156
cts_electricity() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 132
cts_electricity_demand_share() (in module *egon.data.datasets.sanity_checks*), 103
cts_heat() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 132
cts_heat_demand_share() (in module *egon.data.datasets.sanity_checks*), 103
CtsBuildings (class in *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 129
CtsDemandBuildings (class in *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 130
CtsElectricityDemand (class in *egon.data.datasets.electricity_demand*), 126
cutout_heat_demand_germany() (in module *egon.data.datasets.heat_demand*), 153
cyclic_state_of_charge (*EgonPfHvStorage* attribute), 82

D

daily_demand_share (*EgonDailyHeatDemandPerClimateZone* attribute), 154
daily_demand_shares_per_climate_zone() (in module *egon.data.datasets.heat_demand_timeseries.daily*), 155
data_export() (in module *egon.data.datasets.DSM_cts_ind*), 66
data_in_boundaries() (in module *egon.data.datasets.demandregio*), 120

DataBundle (class in *egon.data.datasets.data_bundle*), 119
 Dataset (class in *egon.data.datasets*), 213
 datasets() (in module *egon.data.config*), 63
 datasource (*HotmapsIndustrialSites* attribute), 179
 day_of_year (*EgonDailyHeatDemandPerClimateZone* attribute), 154
 dbahn (*EgonEhvSubstation* attribute), 187
 dbahn (*EgonEhvTransferBuses* attribute), 210
 dbahn (*EgonHvmvSubstation* attribute), 187
 dbahn (*EgonHvmvTransferBuses* attribute), 211
 debkg_id (*Vg250Gem* attribute), 111
 debkg_id (*Vg250Lan* attribute), 193
 debkg_id (*Vg250Sta* attribute), 112
 define_DE_crossbording_pipes_geom_eGon100RE (in module *egon.data.datasets.gas_neighbours.eGon100RE*), 147
 define_gas_buses_abroad() (in module *egon.data.datasets.gas_grid*), 89
 define_gas_nodes_list() (in module *egon.data.datasets.gas_grid*), 90
 define_gas_pipeline_list() (in module *egon.data.datasets.gas_grid*), 90
 define_mv_grid_districts() (in module *egon.data.datasets.mv_grid_districts*), 100
 definition (*EgonDemandRegioWz* attribute), 120
 delete_dsm_entries() (in module *egon.data.datasets.DSM_cts_ind*), 66
 delete_heat_peak_loads_100RE() (in module *egon.data.datasets.heat_supply.individual_heating*), 167
 delete_heat_peak_loads_2035() (in module *egon.data.datasets.heat_supply.individual_heating*), 168
 delete_hp_capacity() (in module *egon.data.datasets.heat_supply.individual_heating*), 168
 delete_hp_capacity_100RE() (in module *egon.data.datasets.heat_supply.individual_heating*), 168
 delete_hp_capacity_2035() (in module *egon.data.datasets.heat_supply.individual_heating*), 168
 delete_mvgd_ts() (in module *egon.data.datasets.heat_supply.individual_heating*), 168
 delete_mvgd_ts_100RE() (in module *egon.data.datasets.heat_supply.individual_heating*), 168
 delete_mvgd_ts_2035() (in module *egon.data.datasets.heat_supply.individual_heating*), 168
 delete_old_entries() (in module *egon.data.datasets.industrial_gas_demand*), 94
 delete_previous_gen() (in module *egon.data.datasets.fill_etrago_gen*), 86
 delete_pypsa_eur_sec_csv_file() (in module *egon.data.datasets.heat_supply.individual_heating*), 168
 delete_synthetic_cts_buildings() (in module *egon.data.datasets.electricity_demand_timeseries.cts_building*), 132
 demand (*DemandCurvesOsmIndustryIndividual* attribute), 182
 demand (*DemandCurvesSitesIndustryIndividual* attribute), 183
 demand (*EgonDemandRegioCtsInd* attribute), 119
 demand (*EgonDemandRegioHH* attribute), 119
 demand (*EgonDemandRegioOsmIndElectricity* attribute), 183
 demand (*EgonDemandRegioSitesIndElectricity* attribute), 183
 demand (*EgonDemandRegioZensusElectricity* attribute), 126
 demand (*EgonPetaHeat* attribute), 152
 DemandCurvesOsmIndustry (class in *egon.data.datasets.industry*), 182
 DemandCurvesOsmIndustryIndividual (class in *egon.data.datasets.industry*), 182
 DemandCurvesSitesIndustry (class in *egon.data.datasets.industry*), 182
 DemandCurvesSitesIndustryIndividual (class in *egon.data.datasets.industry*), 183
 DemandRegio (class in *egon.data.datasets.demandregio*), 119
 demands_per_bus() (in module *egon.data.datasets.electricity_demand_etrago*), 75
 demarcation() (in module *egon.data.datasets.district_heating_areas*), 124
 dependencies (*Dataset* attribute), 213
 dependencies (*Model* attribute), 214
 disaggregate_hp_capacity() (in module *egon.data.datasets.heat_supply.individual_heating*), 168
 disaggregate_pv() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 197
 disaggregate_pv_in_mv_grid() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 197
 description (*EgonScenario* attribute), 206
 DestatisZensusPopulationPerHa (class in *egon.data.datasets.zensus_vg250*), 110
 DestatisZensusPopulationPerHaInsideGermany (class in *egon.data.datasets.zensus_vg250*), 110

111 determine_buildings_with_hp_in_mv_grid() *div_list()* (in module *egon.data.datasets.heat_supply.individual_heating*), 67
 168 *dlr()* (in module *egon.data.datasets.calculate_dlr*), 69
 determine_end_of_life_gens() (in module *DLR_Regions()* (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), *egon.data.datasets.calculate_dlr*), 69
 198 *down_time_before* (*EgonPjHvGenerator* attribute), 79
 determine_hp_cap_buildings_eGon100RE() *download()* (in module *egon.data.datasets.heat_supply.individual_heating*), 119
 168 *egon.data.datasets.data_bundle*), 119
 determine_hp_cap_buildings_eGon100RE_per_mvgrid() *download()* (in module *egon.data.datasets.heat_demand_europe*), 93
 168 *download()* (in module *egon.data.datasets.osm*), 185
 determine_hp_cap_buildings_eGon2035_per_mvgrid() *download()* (in module *egon.data.datasets.tyndp*), 110
 169 *download_and_check()* (in module *egon.data.datasets.zensus*), 212
 determine_hp_cap_peak_load_mvgsd_ts_2035() *era5()* (in module *egon.data.datasets.era5*), 76
 169 *era5()* (in module *egon.data.datasets.era5*), 76
 determine_hp_cap_peak_load_mvgsd_ts_pypsa_downloaded() *files()* (in module *egon.data.datasets.vg250*), 212
 169 *download_hotmaps()* (in module *egon.data.datasets.industrial_sites*), 181
 determine_min_hp_cap_buildings_pypsa_eur_sec() *import_industrial_sites()* (in module *egon.data.datasets.industrial_sites*), 181
 169 *import_industrial_sites()* (in module *egon.data.datasets.industrial_sites*), 181
 determine_minimum_hp_capacity_per_building() *industrial_gas_demand()* (in module *egon.data.datasets.industrial_gas_demand*), 94
 169 *industrial_gas_demand()* (in module *egon.data.datasets.industrial_gas_demand*), 94
 disagg_households_power() (in module *egon.data.datasets.demandregio*), 121
 dist_aggregated_mw (*EgonEtragoTimeseriesIndividualHeating* attribute), 157, 166
 dist_aggregated_mw (*EgonTimeseriesDistrictHeating* attribute), 157
 distribute_cts_demands() (in module *egon.data.datasets.electricity_demand*), 127
 district_heating (*EgonChp* attribute), 117
 district_heating() (in module *egon.data.datasets.heat_demand_timeseries*), 158
 district_heating() (in module *egon.data.datasets.heat_supply*), 172
 district_heating_area_id (*EgonChp* attribute), 117
 district_heating_areas() (in module *egon.data.datasets.district_heating_areas*), 124
 district_heating_id (*EgonDistrictHeatingSupply* attribute), 171
 district_heating_input() (in module *egon.data.datasets.scenario_capacities*), 108
 DistrictHeatingAreas (class in *egon.data.datasets.district_heating_areas*), 87
 122 *egon.data.datasets.DSM_cts_ind*), 67
 122 *dlr()* (in module *egon.data.datasets.calculate_dlr*), 69
 122 *DLR_Regions()* (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), *egon.data.datasets.calculate_dlr*), 69
 122 *EgonPjHvGenerator* attribute), 79
 122 *download()* (in module *egon.data.datasets.heat_supply.individual_heating*), 119
 122 *egon.data.datasets.data_bundle*), 119
 122 *download()* (in module *egon.data.datasets.heat_demand_europe*), 93
 122 *download()* (in module *egon.data.datasets.osm*), 185
 122 *download()* (in module *egon.data.datasets.tyndp*), 110
 122 *download_and_check()* (in module *egon.data.datasets.zensus*), 212
 122 *era5()* (in module *egon.data.datasets.era5*), 76
 122 *era5()* (in module *egon.data.datasets.era5*), 76
 122 *files()* (in module *egon.data.datasets.vg250*), 212
 122 *download_hotmaps()* (in module *egon.data.datasets.industrial_sites*), 181
 122 *import_industrial_sites()* (in module *egon.data.datasets.industrial_sites*), 181
 122 *industrial_gas_demand()* (in module *egon.data.datasets.industrial_gas_demand*), 94
 122 *download_mastr_data()* (in module *egon.data.datasets.mastr*), 96
 122 *download_peta5_0_1_heat_demands()* (in module *egon.data.datasets.heat_demand*), 153
 122 *download_pypsa_technology_data()* (in module *egon.data.datasets.scenario_parameters*), 207
 122 *download_SciGRID_gas_data()* (in module *egon.data.datasets.gas_grid*), 90
 122 *download_seenergies()* (in module *egon.data.datasets.industrial_sites*), 181
 122 *download_zensus_misc()* (in module *egon.data.datasets.zensus*), 212
 122 *download_zensus_pop()* (in module *egon.data.datasets.zensus*), 213
 122 *drop_buildings_outside_grids()* (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 198
 122 *drop_buildings_outside_muns()* (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 198
 122 *drop_bus()* (in module *egon.data.datasets.fix_ehv_subnetworks*), 87
 122 *drop_gens_outside_muns()* (in module *egon.data.datasets.fix_ehv_subnetworks*), 87

`egon.data.datasets.power_plants.pv_rooftop_buildings` (in module `egon.data.datasets.power_plants`), 198
`drop_line()` (in module `egon.data.datasets.fix_ehv_subnetworks`), 87
`drop_temp_tables()` (in module `egon.data.datasets.loadarea`), 184
`drop_temp_tables()` (in module `egon.data.datasets.osm_buildings_streets`), 186
`drop_trafo()` (in module `egon.data.datasets.fix_ehv_subnetworks`), 87
`drop_unallocated_gens()` (in module `egon.data.datasets.power_plants.pv_rooftop_buildings`), 198
`dsm_cts_ind()` (in module `egon.data.datasets.DSM_cts_ind`), 67
`dsm_cts_ind_individual()` (in module `egon.data.datasets.DSM_cts_ind`), 67
`dsm_cts_ind_processing()` (in module `egon.data.datasets.DSM_cts_ind`), 68
`DsmPotential` (class in `egon.data.datasets.DSM_cts_ind`), 64

E

`e_cyclic` (*EgonPfHvStore* attribute), 83
`e_initial` (*EgonPfHvStore* attribute), 83
`e_max` (*EgonDemandregioSitesIndElectricityDsmTimeseries* attribute), 64
`e_max` (*EgonEtragoElectricityCtsDsmTimeseries* attribute), 65
`e_max` (*EgonOsmIndLoadCurvesIndividualDsmTimeseries* attribute), 65
`e_max` (*EgonSitesIndLoadCurvesIndividualDsmTimeseries* attribute), 65
`e_max_pu` (*EgonPfHvStore* attribute), 83
`e_max_pu` (*EgonPfHvStoreTimeseries* attribute), 84
`e_min` (*EgonDemandregioSitesIndElectricityDsmTimeseries* attribute), 64
`e_min` (*EgonEtragoElectricityCtsDsmTimeseries* attribute), 65
`e_min` (*EgonOsmIndLoadCurvesIndividualDsmTimeseries* attribute), 65
`e_min` (*EgonSitesIndLoadCurvesIndividualDsmTimeseries* attribute), 65
`e_min_pu` (*EgonPfHvStore* attribute), 83
`e_min_pu` (*EgonPfHvStoreTimeseries* attribute), 84
`e_nom` (*EgonPfHvStore* attribute), 84
`e_nom_extendable` (*EgonPfHvStore* attribute), 84
`e_nom_max` (*EgonPfHvGenerator* attribute), 79
`e_nom_max` (*EgonPfHvStore* attribute), 84
`e_nom_min` (*EgonPfHvStore* attribute), 84
`echo()` (in module `egon.data`), 63
`efficiency` (*EgonPfHvGenerator* attribute), 79
`efficiency` (*EgonPfHvLink* attribute), 81
`efficiency` (*EgonPfHvLinkTimeseries* attribute), 81
`efficiency_dispatch` (*EgonPfHvStorage* attribute), 82
`efficiency_store` (*EgonPfHvStorage* attribute), 82
`egon.data` (module), 63
`egon.data.airflow` (module), 63
`egon.data.cli` (module), 63
`egon.data.config` (module), 63
`egon.data.datasets` (module), 213
`egon.data.datasets.calculate_dlr` (module), 68
`egon.data.datasets.ch4_prod` (module), 69
`egon.data.datasets.ch4_storages` (module), 70
`egon.data.datasets.chp` (module), 116
`egon.data.datasets.chp.match_nep` (module), 113
`egon.data.datasets.chp.small_chp` (module), 114
`egon.data.datasets.chp_etrago` (module), 72
`egon.data.datasets.data_bundle` (module), 119
`egon.data.datasets.database` (module), 72
`egon.data.datasets.demandregio` (module), 119
`egon.data.datasets.demandregio.install_disaggregator` (module), 119
`egon.data.datasets.district_heating_areas` (module), 122
`egon.data.datasets.district_heating_areas.plot` (module), 122
`egon.data.datasets.DSM_cts_ind` (module), 64
`egon.data.datasets.electrical_neighbours` (module), 72
`egon.data.datasets.electricity_demand` (module), 126
`egon.data.datasets.electricity_demand.temporal` (module), 126
`egon.data.datasets.electricity_demand_etrago` (module), 75
`egon.data.datasets.electricity_demand_timeseries` (module), 146
`egon.data.datasets.electricity_demand_timeseries.ct` (module), 127
`egon.data.datasets.electricity_demand_timeseries.hl` (module), 132
`egon.data.datasets.electricity_demand_timeseries.hl` (module), 136
`egon.data.datasets.electricity_demand_timeseries.ma` (module), 145
`egon.data.datasets.electricity_demand_timeseries.to`

(module), 145
 egon.data.datasets.era5 (module), 76
 egon.data.datasets.etrago_helpers (module), 77
 egon.data.datasets.etrago_setup (module), 78
 egon.data.datasets.fill_etrago_gen (module), 86
 egon.data.datasets.fix_ehv_subnetworks (module), 87
 egon.data.datasets.gas_areas (module), 87
 egon.data.datasets.gas_grid (module), 89
 egon.data.datasets.gas_neighbours (module), 151
 egon.data.datasets.gas_neighbours.eGon1000 (module), 147
 egon.data.datasets.gas_neighbours.eGon2035 (module), 148
 egon.data.datasets.gas_neighbours.gas_abroad (module), 151
 egon.data.datasets.generate_voronoi (module), 92
 egon.data.datasets.heat_demand (module), 152
 egon.data.datasets.heat_demand_europe (module), 92
 egon.data.datasets.heat_demand_timeseries (module), 157
 egon.data.datasets.heat_demand_timeseries.daily (module), 154
 egon.data.datasets.heat_demand_timeseries.idp (module), 155
 egon.data.datasets.heat_demand_timeseries.service (module), 156
 egon.data.datasets.heat_etrago (module), 160
 egon.data.datasets.heat_etrago.hts_etrago (module), 159
 egon.data.datasets.heat_etrago.power_to_heat (module), 159
 egon.data.datasets.heat_supply (module), 171
 egon.data.datasets.heat_supply.district_heating (module), 161
 egon.data.datasets.heat_supply.geothermal (module), 163
 egon.data.datasets.heat_supply.individual_heating (module), 163
 egon.data.datasets.hydrogen_etrago (module), 176
 egon.data.datasets.hydrogen_etrago.bus (module), 173
 egon.data.datasets.hydrogen_etrago.h2_grid (module), 174
 egon.data.datasets.hydrogen_etrago.h2_to_ch4 (module), 174
 egon.data.datasets.hydrogen_etrago.power_to_h2 (module), 175
 egon.data.datasets.hydrogen_etrago.storage (module), 176
 egon.data.datasets.industrial_gas_demand (module), 93
 egon.data.datasets.industrial_sites (module), 179
 egon.data.datasets.industry (module), 182
 egon.data.datasets.industry.temporal (module), 181
 egon.data.datasets.loadarea (module), 184
 egon.data.datasets.low_flex_scenario (module), 185
 egon.data.datasets.mastr (module), 96
 egon.data.datasets.mv_grid_districts (module), 97
 egon.data.datasets.osm (module), 185
 egon.data.datasets.osm_buildings_streets (module), 186
 egon.data.datasets.osmtgmod (module), 188
 egon.data.datasets.osmtgmod.substation (module), 187
 egon.data.datasets.power_etrago (module), 189
 egon.data.datasets.power_etrago.match_ocgt (module), 188
 egon.data.datasets.power_plants (module), 190
 egon.data.datasets.power_plants.assign_weather_data (module), 189
 egon.data.datasets.power_plants.conventional (module), 189
 egon.data.datasets.power_plants.mastr (module), 190
 egon.data.datasets.power_plants.pv_ground_mounted (module), 191
 egon.data.datasets.power_plants.pv_rooftop (module), 191
 egon.data.datasets.power_plants.pv_rooftop_building (module), 191
 egon.data.datasets.power_plants.wind_farms (module), 201
 egon.data.datasets.power_plants.wind_offshore (module), 202
 egon.data.datasets.pypsaeursec (module), 204
 egon.data.datasets.re_potential_areas (module), 204
 egon.data.datasets.renewable_feedin (module), 101
 egon.data.datasets.saltcavern (module),

- 205
- `egon.data.datasets.sanity_checks` (module), 103
- `egon.data.datasets.scenario_capacities` (module), 106
- `egon.data.datasets.scenario_parameters` (module), 206
- `egon.data.datasets.scenario_parameters.parameters` (module), 205
- `egon.data.datasets.society_prognosis` (module), 108
- `egon.data.datasets.storages` (module), 209
- `egon.data.datasets.storages.home_batteries` (module), 207
- `egon.data.datasets.storages.pumped_hydro` (module), 208
- `egon.data.datasets.storages_etrago` (module), 210
- `egon.data.datasets.substation` (module), 210
- `egon.data.datasets.substation_voronoi` (module), 109
- `egon.data.datasets.tyndp` (module), 109
- `egon.data.datasets.vg250` (module), 211
- `egon.data.datasets.vg250_mv_grid_districts` (module), 110
- `egon.data.datasets.zensus` (module), 212
- `egon.data.datasets.zensus_mv_grid_districts` (module), 110
- `egon.data.datasets.zensus_vg250` (module), 110
- `egon.data.db` (module), 214
- `egon.data.metadata` (module), 217
- `egon.data.subprocess` (module), 219
- `eGon100_capacities()` (in module `egon.data.datasets.scenario_capacities`), 108
- `egon_building_peak_loads()` (in module `egon.data.datasets.power_plants.pv_rooftop_buildings`), 198
- `Egon_etrago_gen` (class in `egon.data.datasets.fill_etrago_gen`), 86
- `EgonChp` (class in `egon.data.datasets.chp`), 117
- `EgonCtsElectricityDemandBuildingShare` (class in `egon.data.datasets.electricity_demand_timeseries.cts_buildings`), 130
- `EgonCtsHeatDemandBuildingShare` (class in `egon.data.datasets.electricity_demand_timeseries.cts_buildings`), 130
- `EgonDailyHeatDemandPerClimateZone` (class in `egon.data.datasets.heat_demand_timeseries.daily`), 154
- `EgonDemandRegioCtsInd` (class in `egon.data.datasets.demandregio`), 119
- `EgonDemandRegioHH` (class in `egon.data.datasets.demandregio`), 119
- `EgonDemandRegioHouseholds` (class in `egon.data.datasets.demandregio`), 120
- `EgonDemandRegioOsmIndElectricity` (class in `egon.data.datasets.industry`), 183
- `EgonDemandRegioPopulation` (class in `egon.data.datasets.demandregio`), 120
- `EgonDemandRegioSitesIndElectricity` (class in `egon.data.datasets.industry`), 183
- `EgonDemandregioSitesIndElectricityDsmTimeseries` (class in `egon.data.datasets.DSM_cts_ind`), 64
- `EgonDemandRegioTimeseriesCtsInd` (class in `egon.data.datasets.demandregio`), 120
- `EgonDemandRegioWz` (class in `egon.data.datasets.demandregio`), 120
- `EgonDemandRegioZensusElectricity` (class in `egon.data.datasets.electricity_demand`), 126
- `EgonDestatisZensusHouseholdPerHaRefined` (class in `egon.data.datasets.electricity_demand_timeseries.hh_profiles`), 138
- `EgonDistrictHeatingAreas` (class in `egon.data.datasets.district_heating_areas`), 123
- `EgonDistrictHeatingSupply` (class in `egon.data.datasets.heat_supply`), 171
- `EgonEhvSubstation` (class in `egon.data.datasets.osmtgmod.substation`), 187
- `EgonEhvSubstationVoronoi` (class in `egon.data.datasets.substation_voronoi`), 109
- `EgonEhvTransferBuses` (class in `egon.data.datasets.substation`), 210
- `EgonEra5Cells` (class in `egon.data.datasets.era5`), 76
- `EgonEtragoElectricityCts` (class in `egon.data.datasets.electricity_demand.temporal`), 126
- `EgonEtragoElectricityCtsDsmTimeseries` (class in `egon.data.datasets.DSM_cts_ind`), 65
- `EgonEtragoElectricityHouseholds` (class in `egon.data.datasets.electricity_demand_timeseries.hh_profiles`), 138
- `EgonEtragoHeatCts` (class in `egon.data.datasets.heat_demand_timeseries`), 157
- `EgonEtragoTimeseriesIndividualHeating` (class in `egon.data.datasets.heat_demand_timeseries`), 157
- `EgonEtragoTimeseriesIndividualHeating` (class in `egon.data.datasets.heat_supply.individual_heating`), 166
- `EgonHeatTimeseries` (class in `egon.data.datasets.heat_demand_timeseries.idp_pool`), 155

EgonHomeBatteries	(class in <i>egon.data.datasets.storages.home_batteries</i>), 208	EgonPfHvLoad	(class in <i>egon.data.datasets.etrago_setup</i>), 82
EgonHouseholdPrognosis	(class in <i>egon.data.datasets.society_prognosis</i>), 108	EgonPfHvLoadTimeseries	(class in <i>egon.data.datasets.etrago_setup</i>), 82
EgonHpCapacityBuildings	(class in <i>egon.data.datasets.heat_supply.individual_heating</i>), 166	EgonPfHvStorage	(class in <i>egon.data.datasets.etrago_setup</i>), 82
EgonHvmvSubstation	(class in <i>egon.data.datasets.osmtgmod.substation</i>), 187	EgonPfHvStorageTimeseries	(class in <i>egon.data.datasets.etrago_setup</i>), 83
EgonHvmvSubstationVoronoi	(class in <i>egon.data.datasets.substation_voronoi</i>), 109	EgonPfHvStore	(class in <i>egon.data.datasets.etrago_setup</i>), 83
EgonHvmvTransferBuses	(class in <i>egon.data.datasets.substation</i>), 211	EgonPfHvStoreTimeseries	(class in <i>egon.data.datasets.etrago_setup</i>), 84
EgonIndividualHeatingPeakLoads	(class in <i>egon.data.datasets.heat_demand_timeseries</i>), 157	EgonPfHvTempResolution	(class in <i>egon.data.datasets.etrago_setup</i>), 84
EgonIndividualHeatingSupply	(class in <i>egon.data.datasets.heat_supply</i>), 172	EgonPfHvTransformer	(class in <i>egon.data.datasets.etrago_setup</i>), 84
EgonMapZensusClimateZones	(class in <i>egon.data.datasets.heat_demand_timeseries.daily</i>), 154	EgonPfHvTransformerTimeseries	(class in <i>egon.data.datasets.etrago_setup</i>), 85
EgonMapZensusMvgdBuildings	(class in <i>egon.data.datasets.electricity_demand_timeseries.mapping</i>), 145	EgonPopulationPrognosis	(class in <i>egon.data.datasets.society_prognosis</i>), 108
EgonMaStRConventinalWithoutChp	(class in <i>egon.data.datasets.chp</i>), 117	EgonPowerPlantPvRoofBuilding	(class in <i>egon.data.datasets.power_plants.pv_rooftop_buildings</i>), 192
EgonOsmIndLoadCurvesIndividualDsmTimeseries	(class in <i>egon.data.datasets.DSM_cts_ind</i>), 65	EgonPowerPlants	(class in <i>egon.data.datasets.power_plants</i>), 202
EgonPetaHeat	(class in <i>egon.data.datasets.heat_demand</i>), 152	EgonRenewableFeedIn	(class in <i>egon.data.datasets.era5</i>), 76
EgonPfHvBus	(class in <i>egon.data.datasets.etrago_setup</i>), 78	EgonRePotentialAreaPvAgriculture	(class in <i>egon.data.datasets.re_potential_areas</i>), 204
EgonPfHvBusmap	(class in <i>egon.data.datasets.etrago_setup</i>), 78	EgonRePotentialAreaPvRoadRailway	(class in <i>egon.data.datasets.re_potential_areas</i>), 205
EgonPfHvBusTimeseries	(class in <i>egon.data.datasets.etrago_setup</i>), 78	EgonRePotentialAreaWind	(class in <i>egon.data.datasets.re_potential_areas</i>), 205
EgonPfHvCarrier	(class in <i>egon.data.datasets.etrago_setup</i>), 78	EgonScenario	(class in <i>egon.data.datasets.scenario_parameters</i>), 206
EgonPfHvGasVoronoi	(class in <i>egon.data.datasets.gas_areas</i>), 87	EgonScenarioCapacities	(class in <i>egon.data.datasets.scenario_capacities</i>), 106
EgonPfHvGenerator	(class in <i>egon.data.datasets.etrago_setup</i>), 79	EgonSitesIndLoadCurvesIndividualDsmTimeseries	(class in <i>egon.data.datasets.DSM_cts_ind</i>), 65
EgonPfHvGeneratorTimeseries	(class in <i>egon.data.datasets.etrago_setup</i>), 80	EgonStorages	(class in <i>egon.data.datasets.storages</i>), 209
EgonPfHvLine	(class in <i>egon.data.datasets.etrago_setup</i>), 80	EgonTimeseriesDistrictHeating	(class in <i>egon.data.datasets.heat_demand_timeseries</i>), 157
EgonPfHvLineTimeseries	(class in <i>egon.data.datasets.etrago_setup</i>), 81	EinheitMastrNummer	(EgonMaStRConventinal-WithoutChp attribute), 117
EgonPfHvLink	(class in <i>egon.data.datasets.etrago_setup</i>), 81	el_capacity	(EgonChp attribute), 117
EgonPfHvLinkTimeseries	(class in <i>egon.data.datasets.etrago_setup</i>), 81	el_capacity	(EgonMaStRConventinalWithoutChp attribute), 117
		el_capacity	(EgonPowerPlants attribute), 202
		el_capacity	(EgonStorages attribute), 209
		electrical_bus_id	(EgonChp attribute), 117

ElectricalLoadEtrago (class in module *egon.data.datasets.heat_demand_timeseries*), 158
egon.data.datasets.electricity_demand_etrago), 75

ElectricalNeighbours (class in module *egon.data.datasets.heat_supply.individual_heating*), 169
egon.data.datasets.electrical_neighbours), 72

electricity (*EgonMapZensusMygdBuildings* attribute), 145

electricity() (in module *egon.data.datasets.electricity_demand_etrago*), 75

electricity() (in module *egon.data.datasets.scenario_parameters.parameters*), 205

electricity_parameters (*EgonScenario* attribute), 206

electricitydemand_tj (*SeenergiesIndustrialSites* attribute), 180

emissions_eprtr_2014 (*HotmapsIndustrialSites* attribute), 179

emissions_ets_2014 (*HotmapsIndustrialSites* attribute), 179

engine() (in module *egon.data.db*), 216

engine_for (in module *egon.data.db*), 216

epoch (*Model* attribute), 214

etrago_eGon2035_electricity() (in module *egon.data.datasets.sanity_checks*), 103

etrago_eGon2035_gas_abroad() (in module *egon.data.datasets.sanity_checks*), 104

etrago_eGon2035_gas_DE() (in module *egon.data.datasets.sanity_checks*), 103

etrago_eGon2035_heat() (in module *egon.data.datasets.sanity_checks*), 104

EtragoSetup (class in module *egon.data.datasets.etrago_setup*), 85

eu28 (*SeenergiesIndustrialSites* attribute), 180

excess_heat (*SeenergiesIndustrialSites* attribute), 180

excess_heat_100_200C (*HotmapsIndustrialSites* attribute), 179

excess_heat_200_500C (*HotmapsIndustrialSites* attribute), 179

excess_heat_500C (*HotmapsIndustrialSites* attribute), 179

excess_heat_total (*HotmapsIndustrialSites* attribute), 179

execute_sql() (in module *egon.data.db*), 216

execute_sql_script() (in module *egon.data.datasets.loadarea*), 184

execute_sql_script() (in module *egon.data.datasets.osm_buildings_streets*), 186

execute_sql_script() (in module *egon.data.db*), 216

existing_chp_smaller_10mw() (in module *egon.data.datasets.chp.small_chp*), 114

export_etrago_cts_heat_profiles() (in module *egon.data.datasets.heat_demand_timeseries*), 158

export_min_cap_to_csv() (in module *egon.data.datasets.heat_supply.individual_heating*), 169

export_to_db() (in module *egon.data.datasets.electricity_demand_etrago*), 75

export_to_db() (in module *egon.data.datasets.heat_supply.individual_heating*), 169

extendable_batteries() (in module *egon.data.datasets.storages_etrago*), 210

extendable_batteries_per_scenario() (in module *egon.data.datasets.storages_etrago*), 210

extension_BB() (in module *egon.data.datasets.chp*), 118

extension_BE() (in module *egon.data.datasets.chp*), 118

extension_BW() (in module *egon.data.datasets.chp*), 118

extension_BY() (in module *egon.data.datasets.chp*), 118

extension_district_heating() (in module *egon.data.datasets.chp.small_chp*), 114

extension_HB() (in module *egon.data.datasets.chp*), 118

extension_HE() (in module *egon.data.datasets.chp*), 118

extension_HH() (in module *egon.data.datasets.chp*), 118

extension_industrial() (in module *egon.data.datasets.chp.small_chp*), 115

extension_MV() (in module *egon.data.datasets.chp*), 118

extension_NS() (in module *egon.data.datasets.chp*), 118

extension_NW() (in module *egon.data.datasets.chp*), 118

extension_per_federal_state() (in module *egon.data.datasets.chp.small_chp*), 115

extension_RP() (in module *egon.data.datasets.chp*), 118

extension_SH() (in module *egon.data.datasets.chp*), 118

extension_SL() (in module *egon.data.datasets.chp*), 118

extension_SN() (in module *egon.data.datasets.chp*), 118

extension_ST() (in module *egon.data.datasets.chp*), 118

extension_TH() (in module *egon.data.datasets.chp*), 118

extension_to_areas() (in module *egon.data.datasets.chp.small_chp*), 115
 extract() (in module *egon.data.datasets.osmtgmod.substation*), 188
 extract_amenities() (in module *egon.data.datasets.osm_buildings_streets*), 187
 extract_buildings_filtered_amenities() (in module *egon.data.datasets.osm_buildings_streets*), 187
 extract_buildings_w_amenities() (in module *egon.data.datasets.osm_buildings_streets*), 187
 extract_buildings_wo_amenities() (in module *egon.data.datasets.osm_buildings_streets*), 187
 extract_ways() (in module *egon.data.datasets.osm_buildings_streets*), 187

F

factor_2035 (*HouseholdElectricityProfilesInCensus-Cells* attribute), 138
 factor_2050 (*HouseholdElectricityProfilesInCensus-Cells* attribute), 138
 federal_state (*EgonMaStRConventinalWithoutChp* attribute), 117
 federal_state (*NEP2021ConvPowerPlants* attribute), 107
 federal_state_data() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 198
 federal_states_per_weather_cell() (in module *egon.data.datasets.renewable_feedin*), 102
 feedin (*EgonRenewableFeedIn* attribute), 76
 feedin_per_turbine() (in module *egon.data.datasets.renewable_feedin*), 102
 filename (*Vg250* attribute), 212
 fill_etrago_gen_table() (in module *egon.data.datasets.fill_etrago_gen*), 86
 fill_etrago_gen_time_table() (in module *egon.data.datasets.fill_etrago_gen*), 86
 fill_etrago_generators() (in module *egon.data.datasets.fill_etrago_gen*), 86
 filter_buildings() (in module *egon.data.datasets.osm_buildings_streets*), 187
 filter_buildings_residential() (in module *egon.data.datasets.osm_buildings_streets*), 187
 filter_mastr_geometry() (in module *egon.data.datasets.power_plants*), 203
 filter_zensus_misc() (in module *egon.data.datasets.zensus*), 213
 filter_zensus_population() (in module *egon.data.datasets.zensus*), 213
 finalize_bus_insertion() (in module *egon.data.datasets.etrago_helpers*), 77
 find_bus_id() (in module *egon.data.datasets.power_plants.assign_weather_data*), 189
 find_weather_id() (in module *egon.data.datasets.power_plants.assign_weather_data*), 189
 fix_subnetworks() (in module *egon.data.datasets.fix_ehv_subnetworks*), 87
 FixEhvSubnetworks (class in *egon.data.datasets.fix_ehv_subnetworks*), 87
 fk_s3 (*Vg250Gem* attribute), 111
 fk_s3 (*Vg250Lan* attribute), 193
 fk_s3 (*Vg250Sta* attribute), 112
 foreign_dc_lines() (in module *egon.data.datasets.electrical_neighbours*), 74
 frame_to_numeric() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 198
 frequency (*EgonEhvSubstation* attribute), 187
 frequency (*EgonEhvTransferBuses* attribute), 210
 frequency (*EgonHvmvSubstation* attribute), 188
 frequency (*EgonHvmvTransferBuses* attribute), 211
 fuel_demand (*HotmapsIndustrialSites* attribute), 179
 fueldemand_tj (*SeenergiesIndustrialSites* attribute), 180
 future_heat_demand_germany() (in module *egon.data.datasets.heat_demand*), 153

G

g (*EgonPfHvLine* attribute), 80
 g (*EgonPfHvTransformer* attribute), 84
 gas() (in module *egon.data.datasets.scenario_parameters.parameters*), 206
 gas_parameters (*EgonScenario* attribute), 206
 GasAreaseGon100RE (class in *egon.data.datasets.gas_areas*), 88
 GasAreaseGon2035 (class in *egon.data.datasets.gas_areas*), 88
 GasNeighbours (class in *egon.data.datasets.gas_neighbours*), 151
 GasNodesAndPipes (class in *egon.data.datasets.gas_grid*), 89
 gen (*HvmvSubstPerMunicipality* attribute), 98
 gen (*Vg250Gem* attribute), 111
 gen (*Vg250GemClean* attribute), 98

gen (Vg250GemPopulation attribute), 112
 gen (Vg250Lan attribute), 193
 gen (Vg250Sta attribute), 113
 generate_map() (in module
 egon.data.datasets.power_plants.wind_farms),
 201
 generate_mapping_table() (in module
 egon.data.datasets.electricity_demand_timeseries.hh_buildings),
 134
 generate_resource_fields_from_db_table() (in module egon.data.metadata), 217
 generate_resource_fields_from_sqla_model() (in module egon.data.metadata), 218
 generate_synthetic_buildings() (in module
 egon.data.datasets.electricity_demand_timeseries.hh_buildings),
 135
 generate_wind_farms() (in module
 egon.data.datasets.power_plants.wind_farms),
 201
 generator_id (EgonPfHvGenerator attribute), 79
 generator_id (EgonPfHvGeneratorTimeseries attribute), 80
 gens_id (EgonPowerPlantPvRoofBuilding attribute),
 192
 geom (DestatisZensusPopulationPerHa attribute), 110
 geom (DestatisZensusPopulationPerHaInsideGermany
 attribute), 111
 geom (EgonChp attribute), 117
 geom (EgonEhvSubstationVoronoi attribute), 109
 geom (EgonEra5Cells attribute), 76
 geom (EgonHvmvSubstationVoronoi attribute), 109
 geom (EgonPfHvBus attribute), 78
 geom (EgonPfHvGasVoronoi attribute), 88
 geom (EgonPfHvLine attribute), 80
 geom (EgonPfHvLink attribute), 81
 geom (EgonPfHvTransformer attribute), 85
 geom (EgonPowerPlants attribute), 202
 geom (EgonRePotentialAreaPvAgriculture attribute),
 204
 geom (EgonRePotentialAreaPvRoadRailway attribute),
 205
 geom (EgonRePotentialAreaWind attribute), 205
 geom (EgonStorages attribute), 209
 geom (HotmapsIndustrialSites attribute), 179
 geom (IndustrialSites attribute), 179
 geom (MvGridDistricts attribute), 98
 geom (MvGridDistrictsDissolved attribute), 98
 geom (OsmBuildingsFiltered attribute), 192
 geom (OsmPolygonUrban attribute), 184
 geom (SchmidtIndustrialSites attribute), 180
 geom (SeenergiesIndustrialSites attribute), 180
 geom (Vg250GemPopulation attribute), 112
 geom (VoronoiMunicipalityCuts attribute), 99
 geom (VoronoiMunicipalityCutsAssigned attribute), 99
 geom (VoronoiMunicipalityCutsBase attribute), 99
 geom_building (CtsBuildings attribute), 129
 geom_building (OsmBuildingsSynthetic attribute),
 134
 geom_point (DestatisZensusPopulationPerHa attribute), 110
 geom_point (DestatisZensusPopulationPerHaInsideGermany attribute), 111
 geom_point (EgonEra5Cells attribute), 76
 geom_point (OsmBuildingsFiltered attribute), 192
 geom_point (OsmBuildingsSynthetic attribute), 134
 geom_polygon (EgonDistrictHeatingAreas attribute),
 123
 geom_sub (VoronoiMunicipalityCuts attribute), 99
 geom_sub (VoronoiMunicipalityCutsAssigned attribute), 99
 geom_sub (VoronoiMunicipalityCutsBase attribute), 99
 geometry (EgonDistrictHeatingSupply attribute), 172
 geometry (EgonIndividualHeatingSupply attribute),
 172
 geometry (EgonMaStRConventinalWithoutChp attribute), 117
 geometry (HvmvSubstPerMunicipality attribute), 98
 geometry (Vg250Gem attribute), 111
 geometry (Vg250GemClean attribute), 98
 geometry (Vg250Lan attribute), 193
 geometry (Vg250Sta attribute), 113
 get_annual_household_el_demand_cells() (in module egon.data.datasets.electricity_demand),
 127
 get_building_peak_loads() (in module
 egon.data.datasets.electricity_demand_timeseries.hh_buildings),
 135
 get_buildings_with_decentral_heat_demand_in_mv_grid() (in module egon.data.datasets.heat_supply.individual_heating),
 170
 get_cbat_pbat_ratio() (in module
 egon.data.datasets.storages.home_batteries),
 208
 get_cell_demand_metadata_from_db() (in
 module egon.data.datasets.electricity_demand_timeseries.hh_profiles),
 140
 get_cell_demand_profile_ids() (in module
 egon.data.datasets.electricity_demand_timeseries.hh_profiles),
 140
 get_census_households_grid() (in module
 egon.data.datasets.electricity_demand_timeseries.hh_profiles),
 140
 get_census_households_nuts1_raw() (in
 module egon.data.datasets.electricity_demand_timeseries.hh_profiles),
 141
 get_cross_border_buses() (in module
 egon.data.datasets.electrical_neighbours),
 74

```

get_cross_border_lines() (in module get_residential_buildings_with_decentral_heat_demand
    egon.data.datasets.electrical_neighbours), (in module egon.data.datasets.heat_supply.individual_heating),
74 170
get_cts_buildings_with_decentral_heat_demand_residential_grid(heat_profile_ids() (in
    (in module egon.data.datasets.heat_supply.individual_heating), module egon.data.datasets.heat_supply.individual_heating),
170 171
get_cts_electricity_peak_load() (in module get_scaled_profiles_from_db() (in module
    egon.data.datasets.electricity_demand_timeseries.cts_buildings), egon.data.datasets.electricity_demand_timeseries.hh_profiles),
132 142
get_cts_heat_peak_load() (in module get_sector_parameters() (in module
    egon.data.datasets.electricity_demand_timeseries.cts_buildings), egon.data.datasets.scenario_parameters),
132 207
get_daily_demand_share() (in module get_temperature_interval() (IdpProfiles
    egon.data.datasets.heat_supply.individual_heating), method), 155
170
get_daily_profiles() (in module get_total_heat_pump_capacity_of_mv_grid()
    egon.data.datasets.heat_supply.individual_heating), (in module egon.data.datasets.heat_supply.individual_heating),
170 171
get_foreign_bus_id() (in module get_voronoi_geodataframe() (in module
    egon.data.datasets.electrical_neighbours), egon.data.datasets.generate_voronoi), 92
74
get_foreign_gas_bus_id() (in module get_zensus_cells_with_decentral_heat_demand_in_mv_
    egon.data.datasets.gas_neighbours.eGon2035), (in module egon.data.datasets.heat_supply.individual_heating),
149 171
get_heat_peak_demand_per_building() (in module gf (Vg250Gem attribute), 111
    module egon.data.datasets.heat_supply.individual_heating), parameters (EgonScenario attribute), 206
170 gf (Vg250Lan attribute), 193
get_hh_profiles_from_db() (in module gf (Vg250Sta attribute), 113
    egon.data.datasets.electricity_demand_timeseries.hh_profiles), global_settings() (in module
    egon.data.datasets.scenario_parameters.parameters), 206
141 globalid (SeenergiesIndustrialSites attribute), 180
get_houseprofiles_in_census_cells() (in graph (Tasks attribute), 214
    module egon.data.datasets.electricity_demand_timeseries.hh_profiles), egon.data.datasets.electrical_neighbours),
141 74
get_iee_hh_demand_profiles_raw() (in module grid() (in module egon.data.datasets.gas_neighbours.eGon2035),
    egon.data.datasets.electricity_demand_timeseries.hh_profiles), 142
142 grid_districts() (in module
    egon.data.datasets.power_plants.pv_rooftop_buildings), 99
get_load_timeseries() (in module grid_id (DestatisZensusPopulationPerHa attribute),
    egon.data.datasets.electricity_demand_timeseries.hh_profiles), 142
142 grid_id (DestatisZensusPopulationPerHaInsid-
    eGermany attribute), 111
get_location() (in module grid_id (EgonDestatisZensusHouseholdPerHaRefined
    egon.data.datasets.storages.pumped_hydro), attribute), 138
208 grid_id (HouseholdElectricityProfilesInCensusCells
    attribute), 138
get_map_buses() (in module get_scaled_profiles_from_db() (in module
    egon.data.datasets.electrical_neighbours), egon.data.datasets.fill_etrago_gen), 86
74
get_peta_demand() (in module get_scaled_profiles_from_db() (in module
    egon.data.datasets.electricity_demand_timeseries.cts_buildings), egon.data.datasets.fill_etrago_gen), 86
132
get_peta_demand() (in module get_scaled_profiles_from_db() (in module
    egon.data.datasets.heat_supply.individual_heating), egon.data.datasets.fill_etrago_gen), 86
170
get_probability_for_property() (in module H2_CH4_mix_energy_fractions() (in module
    egon.data.datasets.power_plants.pv_rooftop_buildings), egon.data.datasets.hydrogen_etrago.h2_to_ch4),
199 174
h_value() (in module

```

`egon.data.datasets.heat_demand_timeseries.daily`),
 155
`heat (EgonMapZensusMvgdBuildings attribute)`, 145
`heat () (in module egon.data.datasets.scenario_parameters.parameters)`,
 206
`heat_demand_to_db_table () (in module egon.data.datasets.heat_demand)`, 153
`heat_parameters (EgonScenario attribute)`, 206
`heat_pump_cop () (in module egon.data.datasets.renewable_feedin)`, 102
`HeatDemandEurope (class in egon.data.datasets.heat_demand_europe)`,
 92
`HeatDemandImport (class in egon.data.datasets.heat_demand)`, 152
`HeatEtrago (class in egon.data.datasets.heat_etrago)`,
 160
`HeatPumps2035 (class in egon.data.datasets.heat_supply.individual_heating)`,
 166
`HeatPumps2050 (class in egon.data.datasets.heat_supply.individual_heating)`,
 166
`HeatPumpsPypsaEurSec (class in egon.data.datasets.heat_supply.individual_heating)`,
 166
`HeatSupply (class in egon.data.datasets.heat_supply)`,
 172
`HeatTimeSeries (class in egon.data.datasets.heat_demand_timeseries)`,
 157
`hh_10types (EgonDestatisZensusHouseholdPerHaRefined attribute)`, 138
`hh_5types (EgonDestatisZensusHouseholdPerHaRefined attribute)`, 138
`hh_size (EgonDemandRegioHH attribute)`, 119
`hh_size (EgonDemandRegioHouseholds attribute)`,
 120
`hh_type (EgonDestatisZensusHouseholdPerHaRefined attribute)`, 138
`home_batteries_per_scenario () (in module egon.data.datasets.storages)`, 210
`hotmaps_to_postgres () (in module egon.data.datasets.industrial_sites)`, 181
`HotmapsIndustrialSites (class in egon.data.datasets.industrial_sites)`, 179
`household_prognosis_per_year () (in module egon.data.datasets.society_prognosis)`, 109
`HouseholdDemands (class in egon.data.datasets.electricity_demand_timeseries.hh_profiles)`,
 138
`HouseholdElectricityDemand (class in egon.data.datasets.electricity_demand)`, 126
`HouseholdElectricityProfilesInCensusCells`
 138
 (class in `egon.data.datasets.electricity_demand_timeseries.hh_profiles`)
 134
`HouseholdElectricityProfilesOfBuildings`
 134
`households (EgonDemandRegioHouseholds attribute)`, 120
`households (EgonHouseholdPrognosis attribute)`, 108
`houseprofiles_in_census_cells () (in module egon.data.datasets.electricity_demand_timeseries.hh_profiles)`,
 143
`hp_capacity (EgonHpCapacityBuildings attribute)`,
 166
`hts_to_etrago () (in module egon.data.datasets.heat_etrago.hts_etrago)`,
 159
`HtsEtragoTable (class in egon.data.datasets.heat_etrago.hts_etrago)`,
 159
`HvmvSubstPerMunicipality (class in egon.data.datasets.mv_grid_districts)`, 97
`HydrogenBusEtrago (class in egon.data.datasets.hydrogen_etrago)`, 176
`HydrogenGridEtrago (class in egon.data.datasets.hydrogen_etrago)`, 177
`HydrogenMethaneLinkEtrago (class in egon.data.datasets.hydrogen_etrago)`, 177
`HydrogenPowerLinkEtrago (class in egon.data.datasets.hydrogen_etrago)`, 177
`HydrogenStoreEtrago (class in egon.data.datasets.hydrogen_etrago)`, 178

I
`ibz (Vg250Gem attribute)`, 111
`ibz (Vg250Lan attribute)`, 193
`ibz (Vg250Sta attribute)`, 113
`id (CtsBuildings attribute)`, 129
`id (DestatisZensusPopulationPerHa attribute)`, 111
`id (DestatisZensusPopulationPerHaInsideGermany attribute)`, 111
`id (EgonChp attribute)`, 117
`id (EgonDemandRegioOsmIndElectricity attribute)`, 183
`id (EgonDestatisZensusHouseholdPerHaRefined attribute)`, 138
`id (EgonDistrictHeatingAreas attribute)`, 123
`id (EgonEhvSubstationVoronoi attribute)`, 109
`id (EgonHvmvSubstationVoronoi attribute)`, 109
`id (EgonMaStRConventinalWithoutChp attribute)`, 117
`id (EgonPetaHeat attribute)`, 152
`id (EgonPowerPlants attribute)`, 202
`id (EgonRePotentialAreaPvAgriculture attribute)`, 205
`id (EgonRePotentialAreaPvRoadRailway attribute)`, 205
`id (EgonRePotentialAreaWind attribute)`, 205
`id (EgonStorages attribute)`, 209

id (*HouseholdElectricityProfilesOfBuildings* attribute), 134
 id (*HvMvSubstPerMunicipality* attribute), 98
 id (*IeeHouseholdLoadProfiles* attribute), 139
 id (*IndustrialSites* attribute), 179
 id (*MapZensusDistrictHeatingAreas* attribute), 123
 id (*Model* attribute), 214
 id (*MvGridDistrictsDissolved* attribute), 98
 id (*OsmBuildingsFiltered* attribute), 192
 id (*OsmBuildingsSynthetic* attribute), 134
 id (*OsmPolygonUrban* attribute), 184
 id (*SchmidtIndustrialSites* attribute), 180
 id (*Vg250Gem* attribute), 111
 id (*Vg250GemClean* attribute), 98
 id (*Vg250GemPopulation* attribute), 112
 id (*Vg250Lan* attribute), 193
 id (*Vg250Sta* attribute), 113
 id (*VoronoiMunicipalityCuts* attribute), 99
 id (*VoronoiMunicipalityCutsAssigned* attribute), 99
 identify_bus() (in module *egon.data.datasets.industry.temporal*), 182
 identify_voltage_level() (in module *egon.data.datasets.industry.temporal*), 182
 idp_pool_generator() (in module *egon.data.datasets.heat_demand_timeseries.idp_pool*), 155
 IdpProfiles (class in *egon.data.datasets.heat_demand_timeseries.daily*), 155
 IeeHouseholdLoadProfiles (class in *egon.data.datasets.electricity_demand_timeseries.idp_profiles*), 139
 import_ch4_demandTS() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 149
 import_ch4_grid_capacity() (in module *egon.data.datasets.ch4_storages*), 71
 import_cutout() (in module *egon.data.datasets.era5*), 76
 import_gas_generators() (in module *egon.data.datasets.ch4_prod*), 69
 import_installed_ch4_storages() (in module *egon.data.datasets.ch4_storages*), 71
 import_mastr() (in module *egon.data.datasets.power_plants.mastr*), 190
 import_osm_data() (in module *egon.data.datasets.osmtgmod*), 188
 impute_missing_hh_in_populated_cells() (in module *egon.data.datasets.electricity_demand_timeseries.idp_profiles*), 143
 ind_osm_data_import() (in module *egon.data.datasets.DSM_cts_ind*), 68
 ind_osm_data_import_individual() (in module *egon.data.datasets.DSM_cts_ind*), 68
 ind_sites_data_import() (in module *egon.data.datasets.DSM_cts_ind*), 68
 ind_sites_vent_data_import() (in module *egon.data.datasets.DSM_cts_ind*), 68
 ind_sites_vent_data_import_individual() (in module *egon.data.datasets.DSM_cts_ind*), 68
 index (*EgonDistrictHeatingSupply* attribute), 172
 index (*EgonHomeBatteries* attribute), 208
 index (*EgonIndividualHeatingSupply* attribute), 172
 index (*EgonPowerPlantPvRoofBuilding* attribute), 192
 index (*EgonScenarioCapacities* attribute), 106
 index (*NEP2021ConvPowerPlants* attribute), 107
 individual_heating() (in module *egon.data.datasets.heat_supply*), 172
 individual_heating_per_mv_grid() (in module *egon.data.datasets.heat_demand_timeseries*), 158
 individual_heating_per_mv_grid_100() (in module *egon.data.datasets.heat_demand_timeseries*), 158
 individual_heating_per_mv_grid_2035() (in module *egon.data.datasets.heat_demand_timeseries*), 158
 individual_heating_per_mv_grid_tables() (in module *egon.data.datasets.heat_demand_timeseries*), 159
 industrial_demand_distr() (in module *egon.data.datasets.industry*), 183
 industrial_sites_id (*EgonDemandRegioSitesIndElectricity* attribute), 183
 industrial_sites_id (*EgonDemandregioSitesIndElectricityDsmTimeseries* attribute), 64
 IndustrialDemandCurves (class in *egon.data.datasets.industry*), 183
 IndustrialGasDemand (class in *egon.data.datasets.industrial_gas_demand*), 93
 IndustrialGasDemandeGon100RE (class in *egon.data.datasets.industrial_gas_demand*), 93
 IndustrialGasDemandeGon2035 (class in *egon.data.datasets.industrial_gas_demand*), 94
 IndustrialSites (class in *egon.data.datasets.industrial_sites*), 179
 infer_voltage_level() (in module *egon.data.datasets.power_plants.mastr*), 190
 infer_voltage_level() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 199
 inflow (*EgonPfHvStorage* attribute), 82

inflow (*EgonPfHvStorageTimeseries* attribute), 83
 inhabitants_to_households() (in module *egon.data.datasets.electricity_demand.temporal*), 143
 initialise_bus_insertion() (in module *egon.data.datasets.etrago_helpers*), 77
 insert() (in module *egon.data.datasets.chp_etrago*), 72
 insert() (in module *egon.data.datasets.power_plants.pv_ground_mounted*), 191
 insert() (in module *egon.data.datasets.power_plants.wind_farms*), 201
 insert() (in module *egon.data.datasets.power_plants.wind_offshore*), 202
 insert_biomass_chp() (in module *egon.data.datasets.chp*), 118
 insert_biomass_plants() (in module *egon.data.datasets.power_plants*), 203
 insert_buses() (in module *egon.data.datasets.heat_etrago*), 161
 insert_capacities_per_federal_state_nep() (in module *egon.data.datasets.scenario_capacities*), 108
 insert_carriers() (in module *egon.data.datasets.etrago_setup*), 85
 insert_central_direct_heat() (in module *egon.data.datasets.heat_etrago*), 161
 insert_central_gas_boilers() (in module *egon.data.datasets.heat_etrago*), 161
 insert_central_power_to_heat() (in module *egon.data.datasets.heat_etrago.power_to_heat*), 160
 insert_ch4_demand() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 150
 insert_CH4_nodes_list() (in module *egon.data.datasets.gas_grid*), 91
 insert_ch4_storages() (in module *egon.data.datasets.ch4_storages*), 71
 insert_ch4_stores() (in module *egon.data.datasets.ch4_storages*), 71
 insert_chp_egon100re() (in module *egon.data.datasets.chp*), 118
 insert_chp_egon2035() (in module *egon.data.datasets.chp*), 118
 insert_cts_ind() (in module *egon.data.datasets.demandregio*), 121
 insert_cts_ind_demands() (in module *egon.data.datasets.demandregio*), 121
 insert_cts_ind_wz_definitions() (in module *egon.data.datasets.demandregio*), 121
 insert_cts_load() (in module *egon.data.datasets.electricity_demand.temporal*), 143
 insert_data() (in module *egon.data.datasets.re_potential_areas*), 205
 insert_data_nep() (in module *egon.data.datasets.scenario_capacities*), 108
 insert_egon100re() (in module *egon.data.datasets.chp_etrago*), 72
 insert_feedin() (in module *egon.data.datasets.renewable_feedin*), 102
 insert_gas_buses_abroad() (in module *egon.data.datasets.gas_grid*), 91
 insert_gas_data() (in module *egon.data.datasets.gas_grid*), 91
 insert_gas_data_eGon100RE() (in module *egon.data.datasets.gas_grid*), 92
 insert_gas_grid_capacities() (in module *egon.data.datasets.gas_neighbours.gas_abroad*), 151
 insert_gas_neighbours_eGon100RE() (in module *egon.data.datasets.gas_neighbours.eGon100RE*), 147
 insert_gas_pipeline_list() (in module *egon.data.datasets.gas_grid*), 92
 insert_generators() (in module *egon.data.datasets.electrical_neighbours*), 75
 insert_generators() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 150
 insert_H2_buses_from_CH4_grid() (in module *egon.data.datasets.hydrogen_etrago.bus*), 173
 insert_H2_buses_from_saltcavern() (in module *egon.data.datasets.hydrogen_etrago.bus*), 173
 insert_H2_overground_storage() (in module *egon.data.datasets.hydrogen_etrago.storage*), 176
 insert_h2_pipelines() (in module *egon.data.datasets.hydrogen_etrago.h2_grid*), 174
 insert_H2_saltcavern_storage() (in module *egon.data.datasets.hydrogen_etrago.storage*), 176
 insert_H2_storage_eGon100RE() (in module *egon.data.datasets.hydrogen_etrago.storage*), 176
 insert_h2_to_ch4_eGon100RE() (in module *egon.data.datasets.hydrogen_etrago.h2_to_ch4*), 175
 insert_h2_to_ch4_to_h2() (in module

egon.data.datasets.hydrogen_etrigo.h2_to_ch4,
 175
insert_hh_demand() (in module *egon.data.datasets.demandregio*), 121
insert_household_demand() (in module *egon.data.datasets.demandregio*), 121
insert_hydro_biomass() (in module *egon.data.datasets.power_plants*), 203
insert_hydro_plants() (in module *egon.data.datasets.power_plants*), 203
insert_hydrogen_buses() (in module *egon.data.datasets.hydrogen_etrigo.bus*),
 173
insert_hydrogen_buses_eGon100RE() (in
 module *egon.data.datasets.hydrogen_etrigo.bus*), 173
insert_individual_power_to_heat() (in
 module *egon.data.datasets.heat_etrigo.power_to_heat*),
 160
insert_industrial_gas_demand_egon100RE()
 (in module *egon.data.datasets.industrial_gas_demand*),
 94
insert_industrial_gas_demand_egon2035()
 (in module *egon.data.datasets.industrial_gas_demand*),
 95
insert_industrial_gas_demand_time_series()
 (in module *egon.data.datasets.industrial_gas_demand*),
 95
insert_large_chp() (in module *egon.data.datasets.chp.match_nep*), 113
insert_mastr_chp() (in module *egon.data.datasets.chp.small_chp*), 116
insert_nep_list_powerplants() (in module
egon.data.datasets.scenario_capacities), 108
insert_new_entries() (in module
egon.data.datasets.industrial_gas_demand),
 95
insert_ocgt_abroad() (in module
egon.data.datasets.gas_neighbours.eGon2035),
 150
insert_open_cycle_gas_turbines() (in mod-
 ule *egon.data.datasets.power_etrigo.match_ocgt*),
 188
insert_osm_ind_load() (in module
egon.data.datasets.industry.temporal), 182
insert_PHES() (in module
egon.data.datasets.storages_etrigo), 210
insert_power_to_h2_demand() (in module
egon.data.datasets.gas_neighbours.eGon2035),
 150
insert_power_to_h2_to_power() (in module
egon.data.datasets.hydrogen_etrigo.power_to_h2),
 175
insert_power_to_h2_to_power_eGon100RE()
 (in module *egon.data.datasets.hydrogen_etrigo.power_to_h2*),
 175
insert_power_to_heat_per_level() (in mod-
 ule *egon.data.datasets.heat_etrigo.power_to_heat*),
 160
insert_rural_gas_boilers() (in module
egon.data.datasets.heat_etrigo), 161
insert_scenarios() (in module
egon.data.datasets.scenario_parameters),
 207
insert_sites_ind_load() (in module
egon.data.datasets.industry.temporal), 182
insert_society_data() (in module
egon.data.datasets.demandregio), 121
insert_storage() (in module
egon.data.datasets.electrical_neighbours),
 75
insert_storage() (in module
egon.data.datasets.gas_neighbours.eGon2035),
 150
insert_store() (in module
egon.data.datasets.heat_etrigo), 161
insert_timeseries_per_wz() (in module
egon.data.datasets.demandregio), 121
insert_weather_cells() (in module
egon.data.datasets.era5), 77
include_germany() (in module
egon.data.datasets.zensus_vg250), 113
is_hole (*HvmvSubstPerMunicipality* attribute), 98
is_hole (*Vg250GemClean* attribute), 98
isfloat() (in module
egon.data.datasets.power_plants.mastr),
 190

L
landkreis_number (*SchmidtIndustrialSites* at-
 tribute), 180
lat (*EgonEhvSubstation* attribute), 187
lat (*EgonEhvTransferBuses* attribute), 210
lat (*EgonHvmvSubstation* attribute), 188
lat (*EgonHvmvTransferBuses* attribute), 211
lat (*SchmidtIndustrialSites* attribute), 180
lat (*SeenergiesIndustrialSites* attribute), 180
length (*EgonPfHvLine* attribute), 80
length (*EgonPfHvLink* attribute), 81
level_1_pj (*SeenergiesIndustrialSites* attribute), 180
level_1_r_pj (*SeenergiesIndustrialSites* attribute),
 180
level_1_r_tj (*SeenergiesIndustrialSites* attribute),
 180
level_1_tj (*SeenergiesIndustrialSites* attribute), 180
level_2_pj (*SeenergiesIndustrialSites* attribute), 180
level_2_r_pj (*SeenergiesIndustrialSites* attribute),
 180

level_2_r_tj (*SeenergiesIndustrialSites* attribute), 180
 level_2_tj (*SeenergiesIndustrialSites* attribute), 180
 level_3_pj (*SeenergiesIndustrialSites* attribute), 180
 level_3_r_pj (*SeenergiesIndustrialSites* attribute), 180
 level_3_r_tj (*SeenergiesIndustrialSites* attribute), 180
 level_3_tj (*SeenergiesIndustrialSites* attribute), 180
 license_ccby() (in module *egon.data.metadata*), 218
 license_geonutzv() (in module *egon.data.metadata*), 218
 license_odbl() (in module *egon.data.metadata*), 218
 licenses_datenlizenz_deutschland() (in module *egon.data.metadata*), 218
 lifetime (*EgonPfHvGenerator* attribute), 79
 lifetime (*EgonPfHvLine* attribute), 80
 lifetime (*EgonPfHvLink* attribute), 81
 lifetime (*EgonPfHvStorage* attribute), 82
 lifetime (*EgonPfHvStore* attribute), 84
 lifetime (*EgonPfHvTransformer* attribute), 85
 line_id (*EgonPfHvLine* attribute), 80
 line_id (*EgonPfHvLineTimeseries* attribute), 81
 link_geom_from_buses() (in module *egon.data.datasets.etrango_setup*), 86
 link_id (*EgonPfHvLink* attribute), 81
 link_id (*EgonPfHvLinkTimeseries* attribute), 81
 load_biogas_generators() (in module *egon.data.datasets.ch4_prod*), 70
 load_building_data() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 199
 load_census_data() (in module *egon.data.datasets.district_heating_areas*), 125
 load_curve (*EgonDemandRegioTimeseriesCtsInd* attribute), 120
 load_heat_demands() (in module *egon.data.datasets.district_heating_areas*), 125
 load_id (*EgonPfHvLoad* attribute), 82
 load_id (*EgonPfHvLoadTimeseries* attribute), 82
 load_in_wh (*IeeHouseholdLoadProfiles* attribute), 139
 load_mastr_data() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 199
 load_NG_generators() (in module *egon.data.datasets.ch4_prod*), 70
 load_tables() (in module *egon.data.datasets.fill_etrango_gen*), 86
 LoadArea (class in *egon.data.datasets.loadarea*), 184
 loadareas_add_demand_cts() (in module *egon.data.datasets.loadarea*), 184
 loadareas_add_demand_hh() (in module *egon.data.datasets.loadarea*), 184
 loadareas_add_demand_ind() (in module *egon.data.datasets.loadarea*), 184
 loadareas_create() (in module *egon.data.datasets.loadarea*), 184
 location (*HotmapsIndustrialSites* attribute), 179
 lon (*EgonEhvSubstation* attribute), 187
 lon (*EgonEhvTransferBuses* attribute), 210
 lon (*EgonHvmvSubstation* attribute), 188
 lon (*EgonHvmvTransferBuses* attribute), 211
 lon (*SchmidtIndustrialSites* attribute), 180
 lon (*SeenergiesIndustrialSites* attribute), 180
 LowFlexScenario (class in *egon.data.datasets.low_flex_scenario*), 185

M

main() (in module *egon.data.cli*), 63
 map_all_used_buildings() (in module *egon.data.datasets.electricity_demand_timeseries.mapping*), 145
 map_buses() (in module *egon.data.datasets.hydrogen_etrango.power_to_h2*), 175
 map_buses() (in module *egon.data.datasets.power_etrango.match_ocgt*), 188
 map_carrier() (in module *egon.data.datasets.scenario_capacities*), 108
 map_carriers_tyndp() (in module *egon.data.datasets.electrical_neighbours*), 75
 map_climate_zones_to_zensus() (in module *egon.data.datasets.heat_demand_timeseries.daily*), 155
 map_houseprofiles_to_buildings() (in module *egon.data.datasets.electricity_demand_timeseries.hh_building*), 135
 map_nuts3() (in module *egon.data.datasets.industrial_sites*), 181
 map_zensus_vg250() (in module *egon.data.datasets.zensus_vg250*), 113
 MapMvgriddistrictsVg250 (class in *egon.data.datasets.vg250_mv_grid_districts*), 110
 mapping() (in module *egon.data.datasets.vg250_mv_grid_districts*), 110
 mapping() (in module *egon.data.datasets.zensus_mv_grid_districts*), 110

mapping_zensus_weather() (in module *egon.data.datasets.renewable_feedin*), 102
 MapZensusDistrictHeatingAreas (class in *egon.data.datasets.district_heating_areas*), 123
 MapZensusGridDistricts (class in *egon.data.datasets.zensus_mv_grid_districts*), 110
 MapZensusVg250 (class in *egon.data.datasets.zensus_vg250*), 111
 MapZensusWeatherCell (class in *egon.data.datasets.renewable_feedin*), 101
 marginal_cost (*EgonPfHvGenerator* attribute), 79
 marginal_cost (*EgonPfHvGeneratorTimeseries* attribute), 80
 marginal_cost (*EgonPfHvLink* attribute), 81
 marginal_cost (*EgonPfHvLinkTimeseries* attribute), 82
 marginal_cost (*EgonPfHvStorage* attribute), 82
 marginal_cost (*EgonPfHvStorageTimeseries* attribute), 83
 marginal_cost (*EgonPfHvStore* attribute), 84
 marginal_cost (*EgonPfHvStoreTimeseries* attribute), 84
 mastr_data() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 199
 match_nep_chp() (in module *egon.data.datasets.chp.match_nep*), 114
 match_nep_no_chp() (in module *egon.data.datasets.power_plants.conventional*), 189
 match_nuts3_bl() (in module *egon.data.datasets.demandregio*), 122
 match_osm_and_zensus_data() (in module *egon.data.datasets.electricity_demand_timeseries.hh_buildings*), 135
 match_storage_units() (in module *egon.data.datasets.storages.pumped_hydro*), 208
 max_hours (*EgonPfHvStorage* attribute), 82
 mean_load_factor_per_cap_range() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 200
 merge_inputs() (in module *egon.data.datasets.industrial_sites*), 181
 merge_polygons_to_grid_district() (in module *egon.data.datasets.mv_grid_districts*), 100
 MergeIndustrialSites (class in *egon.data.datasets.industrial_sites*), 179
 meta_metadata() (in module *egon.data.metadata*), 218
 min_down_time (*EgonPfHvGenerator* attribute), 79
 min_up_time (*EgonPfHvGenerator* attribute), 79
 mobility() (in module *egon.data.datasets.scenario_parameters.parameters*), 206
 mobility_parameters (*EgonScenario* attribute), 206
 Model (class in *egon.data.datasets*), 214
 model (*EgonPfHvTransformer* attribute), 85
 modify_tables() (in module *egon.data.datasets.osm*), 185
 municipality_data() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 200
 municipality_id (*VoronoiMunicipalityCuts* attribute), 99
 municipality_id (*VoronoiMunicipalityCutsAssigned* attribute), 99
 municipality_id (*VoronoiMunicipalityCutsBase* attribute), 99
 mv_grid_district_HH_electricity_load() (in module *egon.data.datasets.electricity_demand_timeseries.hh_buildings*), 143
 mv_grid_id (*EgonIndividualHeatingSupply* attribute), 172
 MvGridDistricts (class in *egon.data.datasets.mv_grid_districts*), 98
 MvGridDistrictsDissolved (class in *egon.data.datasets.mv_grid_districts*), 98

N

n_amenities_inside (*CtsBuildings* attribute), 129
 n_amenities_inside (*OsmBuildingsSynthetic* attribute), 134
 name (*Calculate_dlr* attribute), 68
 name (*CH4Production* attribute), 69
 name (*CH4Storages* attribute), 71
 name (*Chp* attribute), 117
 name (*ChpEtrago* attribute), 72
 name (*Dataset* attribute), 214
 name (*DistrictHeatingAreas* attribute), 123
 name (*Egon_etrago_gen* attribute), 86
 name (*EgonPfHvCarrier* attribute), 79
 name (*EgonScenario* attribute), 206
 name (*ElectricalNeighbours* attribute), 73
 name (*FixEhvSubnetworks* attribute), 87
 name (*GasAreaseGon100RE* attribute), 88
 name (*GasAreaseGon2035* attribute), 88
 name (*GasNodesAndPipes* attribute), 89
 name (*HeatDemandEurope* attribute), 93
 name (*HeatDemandImport* attribute), 152
 name (*HeatEtrago* attribute), 160
 name (*HeatSupply* attribute), 172
 name (*HeatTimeSeries* attribute), 158
 name (*HtsEtragoTable* attribute), 159

- name (*HydrogenBusEtrago* attribute), 177
 name (*HydrogenGridEtrago* attribute), 177
 name (*HydrogenMethaneLinkEtrago* attribute), 177
 name (*HydrogenPowerLinkEtrago* attribute), 178
 name (*HydrogenStoreEtrago* attribute), 178
 name (*IndustrialGasDemand* attribute), 93
 name (*IndustrialGasDemandeGon100RE* attribute), 93
 name (*IndustrialGasDemandeGon2035* attribute), 94
 name (*Model* attribute), 214
 name (*NEP2021ConvPowerPlants* attribute), 107
 name (*OsmBuildingsFiltered* attribute), 192
 name (*OsmPolygonUrban* attribute), 184
 name (*PowerPlants* attribute), 203
 name (*RenewableFeedin* attribute), 102
 name (*SanityChecks* attribute), 103
 name (*ScenarioCapacities* attribute), 107
 name (*ScenarioParameters* attribute), 207
 name (*Tyndp* attribute), 110
 name (*WeatherData* attribute), 76
 name_unit (*NEP2021ConvPowerPlants* attribute), 107
 nbd (*Vg250Gem* attribute), 111
 nbd (*Vg250Lan* attribute), 193
 nbd (*Vg250Sta* attribute), 113
 nearest() (in module *egon.data.datasets.chp*), 118
 nearest_polygon_with_substation() (in module *egon.data.datasets.mv_grid_districts*), 100
 neighbor_reduction() (in module *egon.data.datasets.pypsaeursec*), 204
 NEP2021ConvPowerPlants (class in *egon.data.datasets.scenario_capacities*), 106
 next_etrago_id() (in module *egon.data.db*), 216
 nice_name (*EgonPfHvCarrier* attribute), 79
 num_parallel (*EgonPfHvLine* attribute), 80
 num_parallel (*EgonPfHvTransformer* attribute), 85
 numpy_nan() (in module *egon.data.datasets.fill_etrago_gen*), 86
 nuts (*EgonScenarioCapacities* attribute), 106
 nuts (*HvmvSubstPerMunicipality* attribute), 98
 nuts (*Vg250Gem* attribute), 111
 nuts (*Vg250GemClean* attribute), 98
 nuts (*Vg250GemPopulation* attribute), 112
 nuts (*Vg250Lan* attribute), 193
 nuts (*Vg250Sta* attribute), 113
 nuts1 (*EgonDestatisZensusHouseholdPerHaRefined* attribute), 138
 nuts1 (*HouseholdElectricityProfilesInCensusCells* attribute), 138
 nuts1 (*SeenergiesIndustrialSites* attribute), 180
 nuts3 (*EgonDemandRegioCtsInd* attribute), 119
 nuts3 (*EgonDemandRegioHH* attribute), 119
 nuts3 (*EgonDemandRegioHouseholds* attribute), 120
 nuts3 (*EgonDemandRegioPopulation* attribute), 120
 nuts3 (*EgonDestatisZensusHouseholdPerHaRefined* attribute), 138
 nuts3 (*HouseholdElectricityProfilesInCensusCells* attribute), 139
 nuts3 (*IndustrialSites* attribute), 179
 nuts3 (*SeenergiesIndustrialSites* attribute), 181
 nuts_mapping() (in module *egon.data.datasets.scenario_capacities*), 108
 nuts_mvview() (in module *egon.data.datasets.vg250*), 212
- ## O
- objectid (*SeenergiesIndustrialSites* attribute), 181
 offshore_weather_cells() (in module *egon.data.datasets.renewable_feedin*), 102
 old_id (*HvmvSubstPerMunicipality* attribute), 98
 old_id (*Vg250GemClean* attribute), 98
 OpenCycleGasTurbineEtrago (class in *egon.data.datasets.power_etrago*), 189
 OpenStreetMap (class in *egon.data.datasets.osm*), 185
 operator (*EgonEhvSubstation* attribute), 187
 operator (*EgonEhvTransferBuses* attribute), 210
 operator (*EgonHvmvSubstation* attribute), 188
 operator (*EgonHvmvTransferBuses* attribute), 211
 orientation_primary (*EgonPowerPlantPvRoof-Building* attribute), 192
 orientation_primary_angle (*EgonPowerPlant-PvRoofBuilding* attribute), 192
 orientation_uniform (*EgonPowerPlantPvRoof-Building* attribute), 192
 osm (*EgonMapZensusMvgdBuildings* attribute), 145
 osm_buildings() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 200
 osm_id (*DemandCurvesOsmIndustryIndividual* attribute), 182
 osm_id (*EgonDemandRegioOsmIndElectricity* attribute), 183
 osm_id (*EgonEhvSubstation* attribute), 187
 osm_id (*EgonEhvTransferBuses* attribute), 210
 osm_id (*EgonHvmvSubstation* attribute), 188
 osm_id (*EgonHvmvTransferBuses* attribute), 211
 osm_id (*EgonOsmIndLoadCurvesIndividualDsmTime-series* attribute), 65
 osm_id (*OsmBuildingsFiltered* attribute), 192
 osm_id (*OsmPolygonUrban* attribute), 184
 osm_landuse_census_cells_melt() (in module *egon.data.datasets.loadarea*), 185
 osm_landuse_melt() (in module *egon.data.datasets.loadarea*), 185
 osm_www (*EgonEhvSubstation* attribute), 187
 osm_www (*EgonEhvTransferBuses* attribute), 210

osm_www (*EgonHvmvSubstation attribute*), 188
 osm_www (*EgonHvmvTransferBuses attribute*), 211
 OsmBuildingsFiltered (class in *egon.data.datasets.power_plants.pv_rooftop_buildings*), 192
 OsmBuildingsStreets (class in *egon.data.datasets.osm_buildings_streets*), 186
 OsmBuildingsSynthetic (class in *egon.data.datasets.electricity_demand_timeseries.pv_buildings*), 134
 OsmLanduse (class in *egon.data.datasets.loadarea*), 184
 OsmPolygonUrban (class in *egon.data.datasets.loadarea*), 184
 OsmTgmod (class in *egon.data.datasets.osmtgmod*), 188
 osmtgmod() (in module *egon.data.datasets.osmtgmod*), 188
 overlay_grid_districts_with_counties() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 200
 overwrite_H2_pipeline_share() (in module *egon.data.datasets.pypsaeursec*), 204

P

p_max (*EgonDemandregioSitesIndElectricityDsmTimeseries attribute*), 64
 p_max (*EgonEtragoElectricityCtsDsmTimeseries attribute*), 65
 p_max (*EgonOsmIndLoadCurvesIndividualDsmTimeseries attribute*), 65
 p_max (*EgonSitesIndLoadCurvesIndividualDsmTimeseries attribute*), 65
 p_max_pu (*EgonPfHvGenerator attribute*), 79
 p_max_pu (*EgonPfHvGeneratorTimeseries attribute*), 80
 p_max_pu (*EgonPfHvLink attribute*), 81
 p_max_pu (*EgonPfHvLinkTimeseries attribute*), 82
 p_max_pu (*EgonPfHvStorage attribute*), 83
 p_max_pu (*EgonPfHvStorageTimeseries attribute*), 83
 p_min (*EgonDemandregioSitesIndElectricityDsmTimeseries attribute*), 65
 p_min (*EgonEtragoElectricityCtsDsmTimeseries attribute*), 65
 p_min (*EgonOsmIndLoadCurvesIndividualDsmTimeseries attribute*), 65
 p_min (*EgonSitesIndLoadCurvesIndividualDsmTimeseries attribute*), 65
 p_min_pu (*EgonPfHvGenerator attribute*), 79
 p_min_pu (*EgonPfHvGeneratorTimeseries attribute*), 80
 p_min_pu (*EgonPfHvLink attribute*), 81
 p_min_pu (*EgonPfHvLinkTimeseries attribute*), 82
 p_min_pu (*EgonPfHvStorage attribute*), 83
 p_min_pu (*EgonPfHvStorageTimeseries attribute*), 83
 p_nom (*EgonHomeBatteries attribute*), 208
 p_nom (*EgonPfHvGenerator attribute*), 79
 p_nom (*EgonPfHvLink attribute*), 81
 p_nom (*EgonPfHvStorage attribute*), 83
 p_nom_extendable (*EgonPfHvGenerator attribute*), 79
 p_nom_extendable (*EgonPfHvLink attribute*), 81
 p_nom_extendable (*EgonPfHvStorage attribute*), 83
 p_nom_max (*EgonPfHvLink attribute*), 81
 p_nom_max (*EgonPfHvStorage attribute*), 83
 p_nom_min (*EgonPfHvGenerator attribute*), 79
 p_nom_min (*EgonPfHvLink attribute*), 81
 p_nom_min (*EgonPfHvStorage attribute*), 83
 p_set (*DemandCurvesOsmIndustry attribute*), 182
 p_set (*DemandCurvesOsmIndustryIndividual attribute*), 182
 p_set (*DemandCurvesSitesIndustry attribute*), 183
 p_set (*DemandCurvesSitesIndustryIndividual attribute*), 183
 p_set (*EgonDemandregioSitesIndElectricityDsmTimeseries attribute*), 65
 p_set (*EgonEtragoElectricityCts attribute*), 126
 p_set (*EgonEtragoElectricityCtsDsmTimeseries attribute*), 65
 p_set (*EgonEtragoElectricityHouseholds attribute*), 138
 p_set (*EgonEtragoHeatCts attribute*), 157
 p_set (*EgonOsmIndLoadCurvesIndividualDsmTimeseries attribute*), 65
 p_set (*EgonPfHvGenerator attribute*), 79
 p_set (*EgonPfHvGeneratorTimeseries attribute*), 80
 p_set (*EgonPfHvLink attribute*), 81
 p_set (*EgonPfHvLinkTimeseries attribute*), 82
 p_set (*EgonPfHvLoad attribute*), 82
 p_set (*EgonPfHvLoadTimeseries attribute*), 82
 p_set (*EgonPfHvStorage attribute*), 83
 p_set (*EgonPfHvStorageTimeseries attribute*), 83
 p_set (*EgonPfHvStore attribute*), 84
 p_set (*EgonPfHvStoreTimeseries attribute*), 84
 p_set (*EgonSitesIndLoadCurvesIndividualDsmTimeseries attribute*), 65
 path (*HvmvSubstPerMunicipality attribute*), 98
 path (*Vg250GemClean attribute*), 98
 path_length (*EgonPfHvBusmap attribute*), 78
 paths() (in module *egon.data.config*), 64
 peak_load (*DemandCurvesOsmIndustryIndividual attribute*), 182
 peak_load (*DemandCurvesSitesIndustryIndividual attribute*), 183
 peak_load_in_w (*BuildingElectricityPeakLoads attribute*), 134

peak_load_in_w (*BuildingHeatPeakLoads* attribute),
 129, 166
 phase_shift (*EgonPfHvTransformer* attribute), 85
 place_buildings_with_amenities() (in module
 egon.data.datasets.electricity_demand_timeseries.cts_buildings),
 132
 plant (*SchmidtIndustrialSites* attribute), 180
 plot_heat_density_sorted() (in module
 egon.data.datasets.district_heating_areas.plot),
 122
 plot_heat_supply() (in module
 egon.data.datasets.heat_supply.district_heating),
 162
 plot_heat_supply() (in module
 egon.data.datasets.heat_supply.individual_heating),
 171
 plz (*EgonMaStRConventinalWithoutChp* attribute), 117
 point (*EgonEhvSubstation* attribute), 187
 point (*EgonEhvTransferBuses* attribute), 210
 point (*EgonHvmvSubstation* attribute), 188
 point (*EgonHvmvTransferBuses* attribute), 211
 polygon (*EgonEhvSubstation* attribute), 187
 polygon (*EgonEhvTransferBuses* attribute), 210
 polygon (*EgonHvmvSubstation* attribute), 188
 polygon (*EgonHvmvTransferBuses* attribute), 211
 population (*DestatisZensusPopulationPerHa* at-
 tribute), 111
 population (*DestatisZensusPopulationPerHaInsid-
 eGermany* attribute), 111
 population (*EgonDemandRegioPopulation* attribute),
 120
 population (*EgonPopulationPrognosis* attribute), 108
 population_density (*Vg250GemPopulation*
 attribute), 112
 population_in_municipalities() (in module
 egon.data.datasets.zensus_vg250), 113
 population_share() (in module
 egon.data.datasets.scenario_capacities),
 108
 population_to_postgres() (in module
 egon.data.datasets.zensus), 213
 population_total (*Vg250GemPopulation* at-
 tribute), 112
 postcode (*NEP2021ConvPowerPlants* attribute), 107
 potential_germany() (in module
 egon.data.datasets.heat_supply.geothermal),
 163
 power_timeser() (in module
 egon.data.datasets.fill_etrago_gen), 87
 power_type (*EgonEhvSubstation* attribute), 187
 power_type (*EgonEhvTransferBuses* attribute), 210
 power_type (*EgonHvmvSubstation* attribute), 188
 power_type (*EgonHvmvTransferBuses* attribute), 211
 PowerPlants (class in
 egon.data.datasets.power_plants), 202
 prefix() (in module *egon.data.datasets*), 214
 preprocessing() (in module
 egon.data.datasets.osm_buildings_streets),
 200
 probabilities() (in module
 egon.data.datasets.power_plants.pv_rooftop_buildings),
 200
 process_nuts1_census_data() (in module
 egon.data.datasets.electricity_demand_timeseries.hh_profiles),
 144
 production (*HotmapsIndustrialSites* attribute), 179
 profile_id (*HouseholdElectricityProfilesOfBuildings*
 attribute), 134
 profile_share (*EgonCtsElectricityDemandBuild-
 ingShare* attribute), 130
 profile_share (*EgonCtsHeatDemandBuildingShare*
 attribute), 130
 proportionate_allocation() (in module
 egon.data.datasets.electricity_demand_timeseries.hh_profiles),
 144
 psql_insert_copy() (in module
 egon.data.datasets.electricity_demand_timeseries.tools),
 145
 pv() (in module *egon.data.datasets.renewable_feedin*),
 102
 pv_rooftop_per_mv_grid() (in module
 egon.data.datasets.power_plants.pv_rooftop),
 191
 pv_rooftop_per_mv_grid_and_scenario() (in module
 egon.data.datasets.power_plants.pv_rooftop),
 191
 pv_rooftop_to_buildings() (in module
 egon.data.datasets.power_plants.pv_rooftop_buildings),
 200
 PypsaEurSec (class in
 egon.data.datasets.pypsaEurSec), 204

Q

q_set (*EgonEtragoElectricityCts* attribute), 126
 q_set (*EgonEtragoElectricityHouseholds* attribute),
 138
 q_set (*EgonPfHvGenerator* attribute), 79
 q_set (*EgonPfHvGeneratorTimeseries* attribute), 80
 q_set (*EgonPfHvLoad* attribute), 82
 q_set (*EgonPfHvLoadTimeseries* attribute), 82
 q_set (*EgonPfHvStorage* attribute), 83
 q_set (*EgonPfHvStorageTimeseries* attribute), 83
 q_set (*EgonPfHvStore* attribute), 84
 q_set (*EgonPfHvStoreTimeseries* attribute), 84

R

r (*EgonPfHvLine* attribute), 80
 r (*EgonPfHvTransformer* attribute), 85

ramp_limit_down (*EgonPfHvGenerator* attribute), 79
 ramp_limit_shut_down (*EgonPfHvGenerator* attribute), 79
 ramp_limit_start_up (*EgonPfHvGenerator* attribute), 79
 ramp_limit_up (*EgonPfHvGenerator* attribute), 79
 random_ints_until_sum() (in module *egon.data.datasets.electricity_demand_timeseries*), 146
 random_point_in_square() (in module *egon.data.datasets.electricity_demand_timeseries*), 146
 read_and_process_demand() (in module *egon.data.datasets.industrial_gas_demand*), 96
 read_costs() (in module *egon.data.datasets.scenario_parameters.parameters*), 206
 read_csv() (in module *egon.data.datasets.scenario_parameters.parameters*), 206
 read_DE_crossbordering_cap_from_pes() (in module *egon.data.datasets.gas_neighbours.eGon100RE*), 148
 read_industrial_demand() (in module *egon.data.datasets.industrial_gas_demand*), 96
 read_LNG_capacities() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 150
 read_network() (in module *egon.data.datasets.pypsaeursec*), 204
 reduce_synthetic_buildings() (in module *egon.data.datasets.electricity_demand_timeseries.hh_buildings*), 136
 ref (*EgonEhvSubstation* attribute), 187
 ref (*EgonEhvTransferBuses* attribute), 210
 ref (*EgonHvmvSubstation* attribute), 188
 ref (*EgonHvmvTransferBuses* attribute), 211
 refine_census_data_at_cell_level() (in module *egon.data.datasets.electricity_demand_timeseries.hh_profiles*), 144
 regroup_nutsl_census_data() (in module *egon.data.datasets.electricity_demand_timeseries.hh_profiles*), 145
 relate_to_schmidt_sites() (in module *egon.data.datasets.DSM_cts_ind*), 68
 remove_double_bus_id() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 132
 remove_isolated_gas_buses() (in module *egon.data.datasets.gas_grid*), 92
 RenewableFeedin (class in *egon.data.datasets.renewable_feedin*), 101
 residential_and_service_demand (*EgonDistrictHeatingAreas* attribute), 123
 residential_electricity_annual_sum() (in module *egon.data.datasets.sanity_checks*), 104
 residential_electricity_hh_refinement() (in module *egon.data.datasets.sanity_checks*), 104
 resolution (*EgonPfHvTempResolution* attribute), 84
 rs (*Vg250Gem* attribute), 111
 rs (*Vg250Lan* attribute), 193
 rs (*Vg250Sta* attribute), 113
 rs_0 (*HvmvSubstPerMunicipality* attribute), 98
 rs_0 (*Vg250Gem* attribute), 112
 rs_0 (*Vg250GemClean* attribute), 98
 rs_0 (*Vg250GemPopulation* attribute), 112
 rs_0 (*Vg250Lan* attribute), 193
 rs_0 (*Vg250Sta* attribute), 113
 run() (in module *egon.data.datasets.fix_ehv_subnetworks*), 87
 run() (in module *egon.data.datasets.osmtgmod*), 188
 run() (in module *egon.data.subprocess*), 219
 run_pypsa_eur_sec() (in module *egon.data.datasets.pypsaeursec*), 204

S

s_max_pu (*EgonPfHvLine* attribute), 80
 s_max_pu (*EgonPfHvLineTimeseries* attribute), 81
 s_max_pu (*EgonPfHvTransformer* attribute), 85
 s_max_pu (*EgonPfHvTransformerTimeseries* attribute), 85
 s_nom (*EgonPfHvLine* attribute), 80
 s_nom (*EgonPfHvTransformer* attribute), 85
 s_nom_extendable (*EgonPfHvLine* attribute), 80
 s_nom_extendable (*EgonPfHvTransformer* attribute), 85
 s_nom_max (*EgonPfHvLine* attribute), 80
 s_nom_max (*EgonPfHvTransformer* attribute), 85
 s_nom_min (*EgonPfHvLine* attribute), 80
 s_nom_min (*EgonPfHvTransformer* attribute), 85
 SaltcavernData (class in *egon.data.datasets.saltcavern*), 205
 sanity_check_CH4_grid() (in module *egon.data.datasets.sanity_checks*), 105
 sanity_check_CH4_stores() (in module *egon.data.datasets.sanity_checks*), 105
 sanity_check_gas_buses() (in module *egon.data.datasets.sanity_checks*), 105
 sanity_check_gas_links() (in module *egon.data.datasets.sanity_checks*), 105
 sanity_check_gas_one_port() (in module *egon.data.datasets.sanity_checks*), 106
 sanity_check_H2_saltcavern_stores() (in module *egon.data.datasets.sanity_checks*), 105

sanitycheck_dsm() (in module *egon.data.datasets.sanity_checks*), 106
 sanitycheck_emobility_mit() (in module *egon.data.datasets.sanity_checks*), 106
 sanitycheck_home_batteries() (in module *egon.data.datasets.sanity_checks*), 106
 sanitycheck_pv_rooftop_buildings() (in module *egon.data.datasets.sanity_checks*), 106
 SanityChecks (class in *egon.data.datasets.sanity_checks*), 103
 scale_prox2now() (in module *egon.data.datasets.power_plants*), 204
 scenario (*BuildingElectricityPeakLoads* attribute), 134
 scenario (*BuildingHeatPeakLoads* attribute), 129, 166
 scenario (*EgonChp* attribute), 117
 scenario (*EgonCtsElectricityDemandBuildingShare* attribute), 130
 scenario (*EgonCtsHeatDemandBuildingShare* attribute), 130
 scenario (*EgonDemandRegioCtsInd* attribute), 119
 scenario (*EgonDemandRegioHH* attribute), 119
 scenario (*EgonDemandRegioOsmIndElectricity* attribute), 183
 scenario (*EgonDemandRegioSitesIndElectricity* attribute), 183
 scenario (*EgonDemandRegioZensusElectricity* attribute), 126
 scenario (*EgonDistrictHeatingAreas* attribute), 123
 scenario (*EgonDistrictHeatingSupply* attribute), 172
 scenario (*EgonEtragoTimeseriesIndividualHeating* attribute), 157, 166
 scenario (*EgonHomeBatteries* attribute), 208
 scenario (*EgonHpCapacityBuildings* attribute), 166
 scenario (*EgonIndividualHeatingPeakLoads* attribute), 157
 scenario (*EgonIndividualHeatingSupply* attribute), 172
 scenario (*EgonPetaHeat* attribute), 152
 scenario (*EgonPowerPlantPvRoofBuilding* attribute), 192
 scenario (*EgonPowerPlants* attribute), 202
 scenario (*EgonStorages* attribute), 209
 scenario (*EgonTimeseriesDistrictHeating* attribute), 157
 scenario (*MapZensusDistrictHeatingAreas* attribute), 123
 scenario_data() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 200
 scenario_data_import() (in module *egon.data.datasets.heat_demand*), 154
 scenario_name (*EgonScenarioCapacities* attribute), 106
 ScenarioCapacities (class in *egon.data.datasets.scenario_capacities*), 107
 ScenarioParameters (class in *egon.data.datasets.scenario_parameters*), 206
 schmidt_to_postgres() (in module *egon.data.datasets.industrial_sites*), 181
 SchmidtIndustrialSites (class in *egon.data.datasets.industrial_sites*), 179
 scn_name (*DemandCurvesOsmIndustry* attribute), 182
 scn_name (*DemandCurvesOsmIndustryIndividual* attribute), 182
 scn_name (*DemandCurvesSitesIndustry* attribute), 183
 scn_name (*DemandCurvesSitesIndustryIndividual* attribute), 183
 scn_name (*EgonDemandregioSitesIndElectricityDsm-Timeseries* attribute), 65
 scn_name (*EgonEtragoElectricityCts* attribute), 126
 scn_name (*EgonEtragoElectricityCtsDsmTimeseries* attribute), 65
 scn_name (*EgonEtragoElectricityHouseholds* attribute), 138
 scn_name (*EgonEtragoHeatCts* attribute), 157
 scn_name (*EgonOsmIndLoadCurvesIndividualDsm-Timeseries* attribute), 65
 scn_name (*EgonPfHvBus* attribute), 78
 scn_name (*EgonPfHvBusmap* attribute), 78
 scn_name (*EgonPfHvBusTimeseries* attribute), 78
 scn_name (*EgonPfHvGasVoronoi* attribute), 88
 scn_name (*EgonPfHvGenerator* attribute), 79
 scn_name (*EgonPfHvGeneratorTimeseries* attribute), 80
 scn_name (*EgonPfHvLine* attribute), 80
 scn_name (*EgonPfHvLineTimeseries* attribute), 81
 scn_name (*EgonPfHvLink* attribute), 81
 scn_name (*EgonPfHvLinkTimeseries* attribute), 82
 scn_name (*EgonPfHvLoad* attribute), 82
 scn_name (*EgonPfHvLoadTimeseries* attribute), 82
 scn_name (*EgonPfHvStorage* attribute), 83
 scn_name (*EgonPfHvStorageTimeseries* attribute), 83
 scn_name (*EgonPfHvStore* attribute), 84
 scn_name (*EgonPfHvStoreTimeseries* attribute), 84
 scn_name (*EgonPfHvTransformer* attribute), 85
 scn_name (*EgonPfHvTransformerTimeseries* attribute), 85
 scn_name (*EgonSitesIndLoadCurvesIndividualDsm-Timeseries* attribute), 65
 sdv_ars (*Vg250Gem* attribute), 112
 sdv_ars (*Vg250Lan* attribute), 193
 sdv_ars (*Vg250Sta* attribute), 113
 sdv_rs (*Vg250Gem* attribute), 112
 sdv_rs (*Vg250Lan* attribute), 193
 sdv_rs (*Vg250Sta* attribute), 113

- sector (*BuildingElectricityPeakLoads* attribute), 134
- sector (*BuildingHeatPeakLoads* attribute), 129, 166
- sector (*EgonDemandRegioWz* attribute), 120
- sector (*EgonDemandRegioZensusElectricity* attribute), 126
- sector (*EgonMapZensusMvgdBldings* attribute), 145
- sector (*EgonPetaHeat* attribute), 152
- sector (*OsmPolygonUrban* attribute), 184
- sector_name (*OsmPolygonUrban* attribute), 184
- seenergies_to_postgres() (in module *egon.data.datasets.industrial_sites*), 181
- SeenergiesIndustrialSites (class in *egon.data.datasets.industrial_sites*), 180
- select() (in module *egon.data.datasets.heat_demand_timeseries.idp_profiles*), 156
- select_bus_id() (in module *egon.data.datasets.fix_ehv_subnetworks*), 87
- select_chp_from_mastr() (in module *egon.data.datasets.chp.match_nep*), 114
- select_chp_from_nep() (in module *egon.data.datasets.chp.match_nep*), 114
- select_cts_buildings() (in module *egon.data.datasets.electricity_demand_timeseries.cts_buildings*), 132
- select_dataframe() (in module *egon.data.db*), 216
- select_district_heating_areas() (in module *egon.data.datasets.heat_supply.district_heating_areas*), 162
- select_geodataframe() (in module *egon.data.db*), 216
- select_geom() (in module *egon.data.datasets.zensus*), 213
- select_high_heat_demands() (in module *egon.data.datasets.district_heating_areas*), 125
- select_mastr_pumped_hydro() (in module *egon.data.datasets.storages.pumped_hydro*), 209
- select_nep_power_plants() (in module *egon.data.datasets.power_plants.conventional*), 189
- select_nep_pumped_hydro() (in module *egon.data.datasets.storages.pumped_hydro*), 209
- select_no_chp_combustion_mastr() (in module *egon.data.datasets.power_plants.conventional*), 189
- select_target() (in module *egon.data.datasets.power_plants*), 204
- selected_idp_profiles (*EgonHeatTimeseries* attribute), 155
- serial (*CtsBuildings* attribute), 129
- session_scope() (in module *egon.data.db*), 217
- session_scoped() (in module *egon.data.db*), 217
- set_multiindex_to_profiles() (in module *egon.data.datasets.electricity_demand_timeseries.hh_profiles*), 145
- set_numexpr_threads() (in module *egon.data.config*), 64
- set_technology_data() (in module *egon.data.datasets.heat_supply.district_heating*), 162
- set_timeseries() (in module *egon.data.datasets.fill_etrago_gen*), 87
- settings() (in module *egon.data.config*), 64
- setup() (in module *egon.data.datasets*), 214
- shutdown() (in module *egon.data.datasets.database*), 72
- shut_down_cost (*EgonPfHvGenerator* attribute), 79
- sign (*EgonPfHvGenerator* attribute), 79
- sign (*EgonPfHvLoad* attribute), 82
- sign (*EgonPfHvStorage* attribute), 83
- sign (*EgonPfHvStore* attribute), 84
- site_id (*DemandCurvesSitesIndustryIndividual* attribute), 183
- site_id (*EgonSitesIndLoadCurvesIndividualDsm-Timeseries* attribute), 65
- site_id (*HotmapsIndustrialSites* attribute), 179
- siteid (*SeenergiesIndustrialSites* attribute), 181
- sitename (*HotmapsIndustrialSites* attribute), 179
- slp (*EgonDemandRegioTimeseriesCtsInd* attribute), 120
- sn_g (*Vg250Gem* attribute), 112
- sn_g (*Vg250Lan* attribute), 193
- sn_g (*Vg250Sta* attribute), 113
- sn_k (*Vg250Gem* attribute), 112
- sn_k (*Vg250Lan* attribute), 193
- sn_k (*Vg250Sta* attribute), 113
- sn_l (*Vg250Gem* attribute), 112
- sn_l (*Vg250Lan* attribute), 193
- sn_l (*Vg250Sta* attribute), 113
- sn_r (*Vg250Gem* attribute), 112
- sn_r (*Vg250Lan* attribute), 193
- sn_r (*Vg250Sta* attribute), 113
- sn_v1 (*Vg250Gem* attribute), 112
- sn_v1 (*Vg250Lan* attribute), 193
- sn_v1 (*Vg250Sta* attribute), 113
- sn_v2 (*Vg250Gem* attribute), 112
- sn_v2 (*Vg250Lan* attribute), 193
- sn_v2 (*Vg250Sta* attribute), 113
- SocietyPrognosis (class in *egon.data.datasets.society_prognosis*), 109
- solar_thermal() (in module *egon.data.datasets.renewable_feedin*), 102
- sort_and_qcut_df() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 200
- source (*CtsBuildings* attribute), 130

source_id (*EgonChp* attribute), 117
 source_id (*EgonPowerPlants* attribute), 202
 source_id (*EgonStorages* attribute), 209
 sources (*EgonChp* attribute), 117
 sources (*EgonPowerPlants* attribute), 202
 sources (*EgonStorages* attribute), 209
 specific_int_until_sum() (in module *egon.data.datasets.electricity_demand_timeseries*), 146
 split_multi_substation_municipalities() (in module *egon.data.datasets.mv_grid_districts*), 101
 split_mvgs_into_bulks() (in module *egon.data.datasets.heat_supply.individual_heating*), 171
 standing_loss (*EgonPfHvStorage* attribute), 83
 standing_loss (*EgonPfHvStore* attribute), 84
 start_time (*EgonPfHvTempResolution* attribute), 84
 start_up_cost (*EgonPfHvGenerator* attribute), 79
 state_of_charge_initial (*EgonPfHvStorage* attribute), 83
 state_of_charge_set (*EgonPfHvStorage* attribute), 83
 state_of_charge_set (*EgonPfHvStorageTimeseries* attribute), 83
 status (*EgonEhvSubstation* attribute), 187
 status (*EgonEhvTransferBuses* attribute), 211
 status (*EgonHvmvSubstation* attribute), 188
 status (*EgonHvmvTransferBuses* attribute), 211
 status (*NEP2021ConvPowerPlants* attribute), 107
 storage_id (*EgonPfHvStorage* attribute), 83
 storage_id (*EgonPfHvStorageTimeseries* attribute), 83
 StorageEtrago (class in *egon.data.datasets.storages_etrago*), 210
 Storages (class in *egon.data.datasets.storages*), 209
 store() (in module *egon.data.datasets.heat_etrago*), 161
 store_id (*EgonPfHvStore* attribute), 84
 store_id (*EgonPfHvStoreTimeseries* attribute), 84
 store_national_profiles() (in module *egon.data.datasets.electricity_demand_etrago*), 75
 store_national_profiles() (in module *egon.data.datasets.heat_demand_timeseries*), 159
 study_prospective_district_heating_areas (in module *egon.data.datasets.district_heating_areas*), 125
 submit_comment() (in module *egon.data.db*), 217
 subsector (*HotmapsIndustrialSites* attribute), 179
 subsector (*IndustrialSites* attribute), 179
 subsector (*SeenergiesIndustrialSites* attribute), 181
 subst_count (*HvmvSubstPerMunicipality* attribute), 98
 subst_count (*VoronoiMunicipalityCuts* attribute), 99
 subst_count (*VoronoiMunicipalityCutsAssigned* attribute), 99
 subst_count (*VoronoiMunicipalityCutsBase* attribute), 99
 subst_name (*EgonEhvSubstation* attribute), 187
 subst_name (*EgonEhvTransferBuses* attribute), 211
 subst_name (*EgonHvmvSubstation* attribute), 188
 subst_name (*EgonHvmvTransferBuses* attribute), 211
 substation (*EgonEhvSubstation* attribute), 187
 substation (*EgonEhvTransferBuses* attribute), 211
 substation (*EgonHvmvSubstation* attribute), 188
 substation (*EgonHvmvTransferBuses* attribute), 211
 substation_voronoi() (in module *egon.data.datasets.substation_voronoi*), 109
 SubstationExtraction (class in *egon.data.datasets.substation*), 211
 substations_in_municipalities() (in module *egon.data.datasets.mv_grid_districts*), 101
 SubstationVoronoi (class in *egon.data.datasets.substation_voronoi*), 109
 supply() (in module *egon.data.datasets.heat_etrago*), 161
 synthetic_buildings() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 201

T

tags (*OsmBuildingsFiltered* attribute), 192
 tags (*OsmPolygonUrban* attribute), 184
 tap_position (*EgonPfHvTransformer* attribute), 85
 tap_ratio (*EgonPfHvTransformer* attribute), 85
 tap_side (*EgonPfHvTransformer* attribute), 85
 target (*EgonDemandregioSitesIndElectricityDsmTimeseries* attribute), 65
 target (*EgonEtragoElectricityCtsDsmTimeseries* attribute), 65
 target (*EgonOsmIndLoadCurvesIndividualDsmTimeseries* attribute), 65
 target (*EgonSitesIndLoadCurvesIndividualDsmTimeseries* attribute), 65
 target() (in module *egon.data.datasets.zensus*), 213
 targets (*EgonHomeBatteries* attribute), 208
 Task (in module *egon.data.datasets*), 214
 TaskGraph (in module *egon.data.datasets*), 214
 Tasks (class in *egon.data.datasets*), 214
 tasks (Dataset attribute), 214
 temp_id (*EgonPfHvGeneratorTimeseries* attribute), 80
 temp_id (*EgonPfHvLineTimeseries* attribute), 81
 temp_id (*EgonPfHvLinkTimeseries* attribute), 82
 temp_id (*EgonPfHvLoadTimeseries* attribute), 82
 temp_id (*EgonPfHvStorageTimeseries* attribute), 83
 temp_id (*EgonPfHvStoreTimeseries* attribute), 84

temp_id (*EgonPfHvTempResolution* attribute), 84
 temp_id (*EgonPfHvTransformerTimeseries* attribute), 85
 temp_id (*VoronoiMunicipalityCutsAssigned* attribute), 99
 temp_interval() (in module *egon.data.datasets.heat_demand_timeseries.daily*), 155
 temp_resolution() (in module *egon.data.datasets.etrango_setup*), 86
 temperature_class (*EgonDailyHeatDemandPerClimateZone* attribute), 154
 temperature_classes() (in module *egon.data.datasets.heat_demand_timeseries.daily*), 155
 temperature_classes() (in module *egon.data.datasets.heat_demand_timeseries.idp_tool*), 156
 temperature_profile_extract() (in module *egon.data.datasets.heat_demand_timeseries.daily*), 155
 terrain_factor (*EgonPfHvLine* attribute), 80
 terrain_factor (*EgonPfHvLink* attribute), 81
 th_capacity (*EgonChp* attribute), 117
 timeit() (in module *egon.data.datasets.electricity_demand_timeseries.tools*), 146
 timer_func() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 201
 timeseries_per_wz() (in module *egon.data.datasets.demandregio*), 122
 timesteps (*EgonPfHvTempResolution* attribute), 84
 to_postgres() (in module *egon.data.datasets.osm*), 185
 to_postgres() (in module *egon.data.datasets.saltcavern*), 205
 to_postgres() (in module *egon.data.datasets.vg250*), 212
 to_pypsa() (in module *egon.data.datasets.osmtgmod*), 188
 topo (*EgonPfHvLine* attribute), 80
 topo (*EgonPfHvLink* attribute), 81
 topo (*EgonPfHvTransformer* attribute), 85
 trafo_id (*EgonPfHvTransformer* attribute), 85
 trafo_id (*EgonPfHvTransformerTimeseries* attribute), 85
 transfer_busses() (in module *egon.data.datasets.substation*), 211
 turbine_per_weather_cell() (in module *egon.data.datasets.renewable_feedin*), 103
 Tyndp (class in *egon.data.datasets.tyndp*), 109
 tyndp_demand() (in module *egon.data.datasets.electrical_neighbours*), 75
 tyndp_gas_demand() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 151
 tyndp_gas_generation() (in module *egon.data.datasets.gas_neighbours.eGon2035*), 151
 tyndp_generation() (in module *egon.data.datasets.electrical_neighbours*), 75
 type (*EgonPfHvBus* attribute), 78
 type (*EgonPfHvGenerator* attribute), 79
 type (*EgonPfHvLine* attribute), 80
 type (*EgonPfHvLink* attribute), 81
 type (*EgonPfHvLoad* attribute), 82
 type (*EgonPfHvStorage* attribute), 83
 type (*EgonPfHvStore* attribute), 84
 type (*EgonPfHvTransformer* attribute), 85
 type (*IeeHouseholdLoadProfiles* attribute), 139
 unzip_peta5_0_1_heat_demands() (in module *egon.data.datasets.heat_demand*), 154
 up_time_before (*EgonPfHvGenerator* attribute), 80
 update() (*Dataset* method), 214

U

v_ang_max (*EgonPfHvLine* attribute), 80
 v_ang_max (*EgonPfHvTransformer* attribute), 85
 v_ang_min (*EgonPfHvLine* attribute), 81
 v_ang_min (*EgonPfHvTransformer* attribute), 85
 v_mag_pu_max (*EgonPfHvBus* attribute), 78
 v_mag_pu_min (*EgonPfHvBus* attribute), 78
 v_mag_pu_set (*EgonPfHvBus* attribute), 78
 v_mag_pu_set (*EgonPfHvBusTimeseries* attribute), 78
 v_nom (*EgonPfHvBus* attribute), 78
 v_nom (*EgonPfHvLine* attribute), 81
 validate_output() (in module *egon.data.datasets.power_plants.pv_rooftop_buildings*), 201
 version (*Calculate_dlr* attribute), 68
 version (*CH4Production* attribute), 69
 version (*CH4Storages* attribute), 71
 version (*Chp* attribute), 117
 version (*ChpEtrango* attribute), 72
 version (*Dataset* attribute), 214
 version (*DistrictHeatingAreas* attribute), 123
 version (*Egon_etrango_gen* attribute), 86
 version (*EgonPfHvBusmap* attribute), 78
 version (*ElectricalNeighbours* attribute), 73
 version (*FixEhvSubnetworks* attribute), 87
 version (*GasAreaseGon100RE* attribute), 88
 version (*GasAreaseGon2035* attribute), 88

V

version (*GasNodesAndPipes* attribute), 89
 version (*HeatDemandEurope* attribute), 93
 version (*HeatDemandImport* attribute), 152
 version (*HeatEtrago* attribute), 160
 version (*HeatSupply* attribute), 172
 version (*HeatTimeSeries* attribute), 158
 version (*HtsEtragoTable* attribute), 159
 version (*HydrogenBusEtrago* attribute), 177
 version (*HydrogenGridEtrago* attribute), 177
 version (*HydrogenMethaneLinkEtrago* attribute), 177
 version (*HydrogenPowerLinkEtrago* attribute), 178
 version (*HydrogenStoreEtrago* attribute), 178
 version (*IndustrialGasDemand* attribute), 93
 version (*IndustrialGasDemandeGon100RE* attribute), 94
 version (*IndustrialGasDemandeGon2035* attribute), 94
 version (*Model* attribute), 214
 version (*PowerPlants* attribute), 203
 version (*RenewableFeedin* attribute), 102
 version (*SanityChecks* attribute), 103
 version (*ScenarioCapacities* attribute), 107
 version (*ScenarioParameters* attribute), 207
 version (*Tyndp* attribute), 110
 version (*WeatherData* attribute), 76
 Vg250 (class in *egon.data.datasets.vg250*), 212
 vg250 (*OsmPolygonUrban* attribute), 184
 vg250_lan (*MapMvgriddistrictsVg250* attribute), 110
 vg250_metadata_resources_fields() (in module *egon.data.datasets.vg250*), 212
 vg250_municipality_id (*MapZensusVg250* attribute), 111
 vg250_nuts3 (*MapZensusVg250* attribute), 111
 Vg250Gem (class in *egon.data.datasets.zensus_vg250*), 111
 Vg250GemClean (class in *egon.data.datasets.mv_grid_districts*), 98
 Vg250GemPopulation (class in *egon.data.datasets.zensus_vg250*), 112
 Vg250Lan (class in *egon.data.datasets.power_plants.pv_rooftop_buildings*), 192
 Vg250MvGridDistricts (class in *egon.data.datasets.vg250_mv_grid_districts*), 110
 Vg250Sta (class in *egon.data.datasets.zensus_vg250*), 112
 voltage (*EgonEhvSubstation* attribute), 187
 voltage (*EgonEhvTransferBuses* attribute), 211
 voltage (*EgonHvmvSubstation* attribute), 188
 voltage (*EgonHvmvTransferBuses* attribute), 211
 voltage_level (*BuildingElectricityPeakLoads* attribute), 134
 voltage_level (*DemandCurvesOsmIndustryIndividual* attribute), 182

voltage_level (*DemandCurvesSitesIndustryIndividual* attribute), 183
 voltage_level (*EgonChp* attribute), 117
 voltage_level (*EgonPowerPlantPvRoofBuilding* attribute), 192
 voltage_level (*EgonPowerPlants* attribute), 202
 voltage_level (*EgonStorages* attribute), 209
 voronoi_egon100RE() (in module *egon.data.datasets.gas_areas*), 89
 voronoi_egon2035() (in module *egon.data.datasets.gas_areas*), 89
 voronoi_id (*VoronoiMunicipalityCuts* attribute), 99
 voronoi_id (*VoronoiMunicipalityCutsAssigned* attribute), 99
 voronoi_id (*VoronoiMunicipalityCutsBase* attribute), 99
 VoronoiMunicipalityCuts (class in *egon.data.datasets.mv_grid_districts*), 98
 VoronoiMunicipalityCutsAssigned (class in *egon.data.datasets.mv_grid_districts*), 99
 VoronoiMunicipalityCutsBase (class in *egon.data.datasets.mv_grid_districts*), 99

W

w_id (*EgonEra5Cells* attribute), 76
 w_id (*EgonRenewableFeedIn* attribute), 76
 w_id (*MapZensusWeatherCell* attribute), 101
 w_th (*EgonIndividualHeatingPeakLoads* attribute), 157
 weather_cell_id (*EgonPowerPlantPvRoofBuilding* attribute), 192
 weather_cell_id (*EgonPowerPlants* attribute), 202
 weather_cells_in_germany() (in module *egon.data.datasets.renewable_feedin*), 103
 weather_year (*EgonRenewableFeedIn* attribute), 76
 WeatherData (class in *egon.data.datasets.era5*), 76
 weatherId_and_busId() (in module *egon.data.datasets.power_plants.assign_weather_data*), 189
 wind() (in module *egon.data.datasets.renewable_feedin*), 103
 wind_offshore() (in module *egon.data.datasets.renewable_feedin*), 103
 wind_power_states() (in module *egon.data.datasets.power_plants.wind_farms*), 201
 write_hh_profiles_to_db() (in module *egon.data.datasets.electricity_demand_timeseries.hh_profiles*), 145
 write_power_plants_table() (in module *egon.data.datasets.power_plants.assign_weather_data*), 189
 write_refinded_households_to_db() (in module *egon.data.datasets.electricity_demand_timeseries.hh_profiles*), 145

`write_saltcavern_potential()` (in module `egon.data.datasets.hydrogen_etrango.storage`), 176
`write_table_to_postgis()` (in module `egon.data.datasets.electricity_demand_timeseries.tools`), 146
`write_table_to_postgres()` (in module `egon.data.datasets.electricity_demand_timeseries.tools`), 146
`wsk` (`Vg250Gem` attribute), 112
`wsk` (`Vg250Lan` attribute), 193
`wsk` (`Vg250Sta` attribute), 113
`wz` (`DemandCurvesSitesIndustry` attribute), 183
`wz` (`DemandCurvesSitesIndustryIndividual` attribute), 183
`wz` (`EgonDemandRegioCtsInd` attribute), 119
`wz` (`EgonDemandRegioOsmIndElectricity` attribute), 183
`wz` (`EgonDemandRegioSitesIndElectricity` attribute), 183
`wz` (`EgonDemandRegioTimeseriesCtsInd` attribute), 120
`wz` (`EgonDemandRegioWz` attribute), 120
`wz` (`HotmapsIndustrialSites` attribute), 179
`wz` (`IndustrialSites` attribute), 179
`wz` (`SchmidtIndustrialSites` attribute), 180
`wz` (`SeenergiesIndustrialSites` attribute), 181

X

`x` (`EgonPfHvBus` attribute), 78
`x` (`EgonPfHvLine` attribute), 81
`x` (`EgonPfHvTransformer` attribute), 85
`x_mp` (`DestatisZensusPopulationPerHa` attribute), 111

Y

`y` (`EgonPfHvBus` attribute), 78
`y_mp` (`DestatisZensusPopulationPerHa` attribute), 111
`year` (`EgonDemandRegioCtsInd` attribute), 119
`year` (`EgonDemandRegioHH` attribute), 119
`year` (`EgonDemandRegioHouseholds` attribute), 120
`year` (`EgonDemandRegioPopulation` attribute), 120
`year` (`EgonDemandRegioTimeseriesCtsInd` attribute), 120
`year` (`EgonHouseholdPrognosis` attribute), 108
`year` (`EgonPopulationPrognosis` attribute), 108

Z

`zensus_geom` (`MapZensusVg250` attribute), 111
`zensus_household()` (in module `egon.data.datasets.society_prognosis`), 109
`zensus_misc_to_postgres()` (in module `egon.data.datasets.zensus`), 213
`zensus_population()` (in module `egon.data.datasets.society_prognosis`), 109
`zensus_population_id` (`CtsBuildings` attribute), 130
`zensus_population_id` (`EgonDemandRegioZensusElectricity` attribute), 126
`zensus_population_id` (`EgonHeatTimeseries` attribute), 155
`zensus_population_id` (`EgonHouseholdPrognosis` attribute), 108
`zensus_population_id` (`EgonMapZensusClimateZones` attribute), 155
`zensus_population_id` (`EgonMapZensusMygdBuildings` attribute), 145
`zensus_population_id` (`EgonPetaHeat` attribute), 152
`zensus_population_id` (`EgonPopulationPrognosis` attribute), 108
`zensus_population_id` (`MapZensusDistrictHeatingAreas` attribute), 123
`zensus_population_id` (`MapZensusGridDistricts` attribute), 110
`zensus_population_id` (`MapZensusVg250` attribute), 111
`zensus_population_id` (`MapZensusWeatherCell` attribute), 101
`ZensusMiscellaneous` (class in `egon.data.datasets.zensus`), 212
`ZensusMvGridDistricts` (class in `egon.data.datasets.zensus_mv_grid_districts`), 110
`ZensusPopulation` (class in `egon.data.datasets.zensus`), 212
`ZensusVg250` (class in `egon.data.datasets.zensus_vg250`), 113
`zip_and_municipality_from_standort()` (in module `egon.data.datasets.power_plants.mastr`), 190