reegis Documentation

CC BY 4.0 - Uwe Krien

Mar 25, 2021

Contents

The reegis repository provides tools to fetch, prepare and organise input data for heat and power models. At the moment the focus is on the territory of Germany but some tools can be used for european models as well.

CHAPTER 1

Installation

On a Linux Debian system you can use the following command to solve all requirements beforehand.

For other Linux systems you may have to adapt the package names. For Windows systems you can just start pip install and fix occurring errors step by step.

The reegis library is designed for Python 3 and tested on Python >= 3.6. We highly recommend to use virtual environments. Please see the installation page of the oemof documentation for complete instructions on how to install python and a virtual environment on your operating system.

If you have a working Python 3 environment, use pypi to install the latest reegis version:

CHAPTER 2

Documentation

The reegis documentation is powered by readthedocs.

Go to the download page to download different versions and formats (pdf, html, epub) of the documentation.

CHAPTER 3

Contributing

We are warmly welcoming all who want to contribute to the reegis library. If you frequently use one of the data sources you may contact me and help to maintain the packages. Don't be shy, even if you are a beginner.

CHAPTER 4

Citing reegis

Go to the Zenodo page of reegis to find the DOI of your version. To cite all reegis versions use:

CHAPTER 5

License

Copyright (c) 2019 Uwe Krien, nesnoj

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 6

Basic usage information

There are two types of data functions in reegis. The high level functions contain *by_region* functions, that make it possible to get data for a specific region set. Basic functions provide useful functions to get a pandas.DataFrame from a specific data source. These functions may return more or less raw data. Using pandas.DataFrame it is still pretty easy to process these tables to your own needs.

The region set used in the following examples is the federal state set. This set contains 17 regions (16 federal states plus one offshore region).

See the `fig_federal_states_polygons()` to get the full code of this figure.

CHAPTER 7

Important usage note

Some functions may take some minutes even hours on the first run. Calculations that need a lot of time will store the result on your hard disc so that the next run will be a lot shorter on the same computer.

Use a logger to see the progress:

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

CHAPTER 8

Spatial data functions

8.1 Power plants

The powerplant module is based on OPSD.

Get the capacity of powerplants for given regions and a specific year. Use the pandas functions to process the result. The code below will show a typical plot with the capacity for every federal state by fuel for the year 2014.

```
from matplotlib import pyplot as plt
from reegis import powerplants, geometries as geo
geometries = geo.get_federal_states_polygon()
year = 2014
my_pp = powerplants.get_powerplants_by_region(
    geometries, year, 'federal_states')
column = 'capacity_{0}'.format(year)
my_pp[column].unstack().plot(kind='bar', stacked=True)
plt.show()
```

To validate the function the results have been compared to data from the Federal Network Agency (BNetzA). The following plot shows the results of reegis on the left and the data from the BNetzA on the right. Some of the differences between reegis and BNetzA are caused by the different methods. Power plants that were put into operation in the given year are taken only partly into account, proportional to the running time in the given year. Typically renewable power plants are mainly build in the second have of the year. The BNetzA take all powerplants into account that were build until a specific dead line. For an energy model of the year 2014 the reegis method makes more sense.

The full code of the plot can be found here `fig_powerplants()`.

For the full API see `powerplants` .

8.2 Inhabitants

The inhabitants data come from the Federal Agency for Cartography and Geodesy (BKG)

Inhabitants date is available for about 11.400 municipalities in Germany. To get the number of inhabitants for a polygon a map of centroids of these municipalities is used and summed up within each polygon.

```
from reegis import geometries, inhabitants
fs=geometries.get_federal_states_polygon()
inhabitants.get_inhabitants_by_region(2014, fs, name='federal_states').sum()
```

For the full API see *inhabitants*.

8.3 openEgo - spatial contribution of annual electricity demand

The approach is based on the openEgo project.

This package will download about 1.2 GB of data. This will take a while on the first run depending on your internet connection.

The openEgo module will return the absolute demand for more than 200.000 regions. That makes it easy to sum up the results for a given region polygon.

The openEgo data set is not available for different years so it is recommended to use them for spatial contribution and scale it with the overall annual demand of Germany (see *Energy data from the energy ministry (BMWi)*).

```
from reegis import openego, geometries

federal_states=geometries.get_federal_states_polygon()
ego_demand=openego.get_ego_demand_by_region(
    federal_states, 'federal_states', grouped=True)

# the share of the overall demand
share=ego_demand.div(ego_demand.sum())
print(share.mul(100).round(1))  # percentage

# the scaled overall demand (eg. 555 TWh)
print(share.mul(555))
```

For the federal states it is also possible to get the electricity demand from the energy balance. We use this to validate the openego method.

The full code of the plot can be found here `fig_electricity_demand_by_state()`.

For the full API see *openego*.

8.4 Electricity demand

The electricity profile is taken from the *Demand profile from ENTSO-E*, the spatial distribution of the *openEgo - spatial contribution of annual electricity demand* is used.

The annual demand is either taken from BMWi (see: *Energy data from the energy ministry (BMWi)*), openEgo (see: *openEgo - spatial contribution of annual electricity demand*), entso (see. *Demand profile from ENTSO-E*) or can be passed by the user.

```
from reegis import demand_elec, geometries
fs=geometries.get_federal_states_polygon()
```

(continues on next page)

(continued from previous page)

```
annual_demand='bmwi'
my_profile=demand_elec.get_entsoe_profile_by_region(
    fs, 2014, 'test', annual_demand)
```

The full code of the plot can be found here `fig_electricity_profile_from_entsoe()`.

```
from reegis import demand_elec, geometries
fs=geometries.get_federal_states_polygon()

p1=demand_elec.get_entsoe_profile_by_region(fs, 2014, 'test', 'entsoe')
p['entsoe']=p1.sum().sum()

p2=demand_elec.get_entsoe_profile_by_region(fs, 2013, 'test', 'bmwi')
p['bmwi']=p2.sum().sum()

p3=demand_elec.get_entsoe_profile_by_region(fs, 2013, 'test', 'openego')
p['openego']=p3.sum().sum()

p4=demand_elec.get_entsoe_profile_by_region(fs, 2011, 'test', 555555)
p['user value']=p4.sum().sum()
```

The full code of the plot can be found here `fig_scaled_electricity_profile()`.

For the full API see `demand_elec` .

8.4.1 Heat demand

The heat demand is based on the energy balance of the federal states.

For the full API see `demand_heat` .

8.4.2 Feedin time series

At the moment feed-in time series are calculated using the HZG coastdat2 weather data set. This data set is deprecated and will be replaced by the HZG OpenFred data set using the feedinlib.

The feed-in calculations are using the windpowerlib and the pvlib.

For the full API see *`feedin`* .

CHAPTER 9

Static data functions

9.1 Energy Balance

Get the energy balance of the federal states for a given year. The data is taken from a csv-file that is manually downloaded from the LAK page.

https://www.lak-energiebilanzen.de/eingabe-dynamisch/?a=e900

As an automatic download does not work you may have to download the file to get the latest updates. Just rename the downloaded file to *energy_balance_federal_states.csv* and replace the existing file in the *data/static/* directory of the reegis package. Alternatively you can download the file and adapt the path in the config file (*energy_balance: energy_balance_states*) or use the config module to set a ne path.

Usage with file in the default directory:

```
from reegis import energy_balance as eb

year=2012
states=['BB', 'NW']
fuel='lignite (raw)'
row='extraction'

my_eb=eb.get_states_energy_balance(year)
print(my_eb.loc[(states, row), fuel])
```

Usage with alternative file:

```
from reegis import energy_balance as eb, config as cfg

year=2012
states=['BB', 'NW']
fuel='lignite (raw)'
row='extraction'
fn='/my/path/my_file.csv'

```

(continues on next page)

(continued from previous page)

```
cfg.tmp_set('energy_balance', 'energy_balance_states', fn)
my_eb=eb.get_states_energy_balance(year)
print(my_eb.loc[(states, row), fuel])
```

If no year is passed to the function the whole table will be returned. This can be used to show changes over the time.

```
from matplotlib import pyplot as plt
from reegis import energy_balance as eb

fuel='lignite (raw)'
ax=plt.figure(figsize=(9, 5)).add_subplot(1, 1, 1)

my_eb=eb.get_states_energy_balance()
my_eb.loc[(slice(None), slice(None), 'extraction'), fuel].groupby(
    level=0).sum().plot(ax=ax)
plt.title("Extraction of raw lignite in Germany")
plt.xlabel('year')
plt.ylabel('energy [TJ]')
plt.ylim(bottom=0)
plt.show()
```

The full code of the plot can be found here `fig_energy_balance_lignite_extraction()`.

The reason for the drop for the last year is not that extraction of raw extraction ended but that the data set for 2016 is not complete yet. So be careful with most recent data sets and check them before use.

If you frequently work with energy balances please contact the author and give your feedback or help to improve and maintain the API.

For the full API see *energy_balance*.

9.2 Energy data from the energy ministry (BMWi)

The ministry of energy BMWI provides an excel sheet which is not optimised for automatic data processing. Nevertheless, it is possible to get some basic data from there. Be careful with updates, because the structure of the sheets may vary in the future.

9.2.1 Annual electricity demand

Fetch the annual electricity demand from 1991 on. In the following code example it is used to create a time series.

```
demand=pd.Series()
ax=plt.figure(figsize=(9, 4)).add_subplot(1, 1, 1)
for year in range(1991, 2016):
    print(year)
    demand.loc[year]=bmwi.get_annual_electricity_demand_bmwi(year)
print(demand)
```

The full code of the plot can be found here `fig_energy_demand_germany_bmwi()`.

9.2.2 Capacity of renewable energy plants

The example shows the capacity of hydro energy plants in Germany in 2016.

```
re=bmwi_re_energy_capacity()
print(re.loc[2016, ('water', 'capacity')])
```

If you frequently work with BMWi data please contact the author and give your feedback or help to improve and maintain the API.

For the full API see *bmwi* .

9.3 Demand profile from ENTSO-E

The electricity profile is taken from ENTSO-E time series provided by OPSD demand time series.

```
entsoe=get_entsoe_load(2015)
```

For the full API see *entsoe* .

CHAPTER 10

API

10.1 reegis.bmwi module

This module is designed to download and prepare BMWi data.

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.bmwi.**bmwi_re_energy_capacity**()

Prepare the energy production and capacity table from sheet 20.

capacity: [MW] energy: [GWh] fraction: [-]

Examples

```
>>> re=bmwi_re_energy_capacity()
>>> int(re.loc[2016, ('water', 'capacity')])
5629
```

reegis.bmwi.**get_annual_electricity_demand_bmwi**(*year*)

Returns the annual demand for the given year from the BMWI Energiedaten in TWh (Tera Watt hours). Will return None if data for the given year is not available.

Examples

```
>>> get_annual_electricity_demand_bmwi(2014)  # puppel
523.988
```

reegis.bmwi.**get_bmwi_energiedaten_file**(*overwrite=False*)

Download BMWi energy data table.

reegis.bmwi.**read_bmwi_sheet_7**(*sub*)

Parameters **sub** (*str*) – Sub-table ‘a’ or ‘b’.

Returns

Return type pd.DataFrame

Examples

```
>>> my_fs = read_bmwi_sheet_7('a').sort_index()
>>> int(float(my_fs.loc[('Industrie', 'gesamt'), 2014]))
2545
>>> my_fs = read_bmwi_sheet_7('b').sort_index()
>>> int(my_fs.loc[('private Haushalte', 'gesamt'), 2014])
2188
```

10.2 reegis.energy_balance module

Prepare parts of the energy balance of Germany and its federal states.

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.energy_balance.**check_transformation_balance**(*years=None*, *balance=None*, *path=None*)

Checks the balance of the transformation balance. If the difference is greater than 5 the name of the region and the difference will be printed. If a path is given wrong balances will be stored as an excel sheet in this path. One excel table for each year will be created.

Parameters

- **years** (*list*) – List of years to check.
- **balance** (*pandas.DataFrame (optional)*) – A valid transformation balance to check.
- **path** (*str*) – A directory where the regions

Examples

```
>>> check_transformation_balance([2014])
2014 - BB: 460589
2014 - BW: 706972
2014 - BY: 2288252
2014 - MV: 19242
2014 - NI: 705997
2014 - SH: 377561
2014 - ST: 51495
>>> ub=get_transformation_balance(2014)
>>> ub=check_transformation_balance(balance=ub)
nn - BB: 460589
nn - BW: 706972
nn - BY: 2288252
nn - MV: 19242
nn - NI: 705997
```

(continues on next page)

(continued from previous page)

```
nn - SH: 377561
nn - ST: 51495
```

reegis.energy_balance.**fix_transformation_balance**(*eb*)
This is a fix after a manual analysis of the energy balances.

Use with care and check the results.,

reegis.energy_balance.**fix_usage_balance**(*eb*, *year*)
Fixes the energy balances after analysing them. This is done manually.

reegis.energy_balance.**get_de_balance**(*year*)
Download and return energy balance of germany for a given year.

reegis.energy_balance.**get_de_usage_balance**(*year*, *grouped=False*)

Parameters

- **year** –
- **grouped** –

Examples

```
>>> df=get_de_usage_balance(2015, True)
>>> df.loc['total', 'total']
8898093
```

reegis.energy_balance.**get_domestic_retail_share**(*year*, *grouped=False*)

Parameters

- **year** –
- **grouped** –

Examples

```
>>> df=get_domestic_retail_share(2014, True)
>>> df.loc['district heating', 'domestic']
0.73
```

reegis.energy_balance.**get_eb_index_translation_dict**()

reegis.energy_balance.**get_states_energy_balance**(*year=None*)
Get the energy balance for a given year. The input file is the csv-file downloaded from: https://www.lak-energiebilanzen.de/eingabe-dynamisch/?a=e900

Parameters **year** (*int or None*) – If year is None all possible years will be returned.

Returns

Return type pandas.DataFrame

Notes

Translation of the index is incomplete.

Examples

```
>>> eb=get_states_energy_balance(2012)
>>> eb.loc[(['BB', 'NW'], 'extraction'), 'lignite (raw)'].round(1)
BB  extraction    316931.2
NW  extraction    927025.0
Name: lignite (raw), dtype: float64
>>> eb=get_states_energy_balance()
>>> eb.loc[([2012, 2013], ['BB', 'NW'], 'extraction'), 'lignite (raw)'
...     ].round(1).sort_index()
2012  BB  extraction    316931.2
      NW  extraction    927025.0
2013  BB  extraction    318703.2
      NW  extraction    894546.0
Name: lignite (raw), dtype: float64
```

reegis.energy_balance.**get_transformation_balance**(*year*)

Reshape the energy balance and return the transformation part as a MultiIndex DataFrame.

Parameters **year** (*int*) –

Returns

Return type pandas.DataFrame

Examples

```
>>> year=2014
>>> ub=get_transformation_balance(year)
>>> int(ub.loc[('BB', 'input', 'Heizwerke'), 'total'])
0
>>> ub=fix_transformation_balance(ub)
>>> int(ub.loc[('BB', 'input', 'Heizwerke'), 'total'])
5347
```

reegis.energy_balance.**get_transformation_balance_by_region**(*regions*, *year*, *name='region'*, *fix=False*)

Get the transformation part of the energy balance for a given region set. The values will be recalculated by the number of inhabitants.

Parameters

- **year** (*int*) –
- **regions** (*GeoDataFrame*) –
- **name** (*str*) –
- **fix** (*bool*) –

Returns

Return type pandas.DataFrame

Examples

```
>>> cb_orig=get_transformation_balance(2014)
>>> regions=geometries.load(
...     cfg.get('paths', 'geometry'),
...     cfg.get('geometry', 'de21_polygons'))
>>> cb=get_transformation_balance_by_region(regions, 2014, 'de21')
>>> int(cb.sum()['electricity']) == int(cb_orig.sum()['electricity'])
True
```

reegis.energy_balance.**get_usage_balance**(*year*, *grouped=False*)

Get the usage part of the energy balance.

Parameters

- **year** (*int*) – Year of the energy balance.
- **grouped** (*bool*) – If set to True the fuels will be grouped to main groups like hard coal or lignite.

Returns

Return type pandas.DataFrame

Examples

```
>>> year=2013
>>> cb=get_usage_balance(year)
>>> total=cb.pop('total')
>>> int((cb.loc['BE'].sum(axis=1) - total.loc['BE']).sum())
0
>>> int((cb.loc['ST'].sum(axis=1) - total.loc['ST']).sum())
-8952
>>> int((cb.loc['BY'].sum(axis=1) - total.loc['BY']).sum())
-17731
>>> cb=get_usage_balance(year)
>>> cb=fix_usage_balance(cb, year)
>>> total=cb.pop('total')
>>> int((cb.loc['BE'].sum(axis=1) - total.loc['BE']).sum())
0
>>> int((cb.loc['ST'].sum(axis=1) - total.loc['ST']).sum())
0
>>> int((cb.loc['BY'].sum(axis=1) - total.loc['BY']).sum())
0
```

10.3 reegis.entsoe module

Download and prepare entsoe load profile from opsd data portal.

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.entsoe.**get_entsoe_load**(*year*, *version=None*)

Parameters

- **year** –

- **version** –

Examples

```
>>> entsoe=get_entsoe_load(2015)
>>> float(round(entsoe.sum()/1e6, 1))
479.5
```

reegis.entsoe.**get_entsoe_renewable_data**(*file=None*, *version=None*)
Load the default file for re time series or a specific file.

Examples

```
>>> my_re=get_entsoe_renewable_data()
>>> int(my_re['DE_solar_generation_actual'].sum())
188160676
```

reegis.entsoe.**get_filtered_file**(*name*, *url*, *version=None*)

reegis.entsoe.**prepare_de_file**(*filename=None*, *overwrite=False*, *version=None*)
Convert demand file. CET index and Germany's load only.

reegis.entsoe.**read_original_timeseries_file**(*orig_csv_file=None*, *overwrite=False*, *version=None*)
Read timeseries file if it exists. Otherwise download it from opsd.

reegis.entsoe.**split_timeseries_file**(*filename=None*, *overwrite=False*, *version=None*)
Split table into load and renewables.

10.4 reegis.feedin module

This module is designed for the use with the pvlib, windpowerlib. If you want to use other libraries you have to adapt the code.

The weather data set has to be a DataFrame with the following columns:

pvlib:

- ghi - global horizontal irradiation [W/m2]
- dni - direct normal irradiation [W/m2]
- dhi - diffuse horizontal irradiation [W/m2]
- temp_air - ambient temperature [°C]

windpowerlib:

- pressure - air pressure [Pa]
- temp_air - ambient temperature [K]
- v_wind - horizontal wind speed [m/s]
- z0 - roughness length [m]

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.feedin.**create_pvlib_sets**()
Create pvlib parameter sets from the solar.ini file.

Returns

Return type dict

Examples

```
>>> pv_set=create_pvlib_sets()['M_LG290G3__I_ABB_MICRO_025_US208'][3]
>>> int(pv_set['surface_azimuth'])
180
>>> for key in sorted(pv_set.keys()):
...     print(key)
albedo
inverter_parameters
module_parameters
name
p_peak
surface_azimuth
surface_tilt
```

reegis.feedin.**create_windpowerlib_sets**()
Create parameter sets for the windpowerlib from wind.ini.

Returns

Return type dict

Examples

```
>>> wind_set=create_windpowerlib_sets()['ENERCON_82_hub98_2300'][1]
>>> wind_set['hub_height']
98
>>> sorted(list(create_windpowerlib_sets().keys()))[:2]
['ENERCON_127_hub135_7500', 'ENERCON_82_hub138_2300']
>>> for key in sorted(wind_set.keys()):
...     print(key)
hub_height
turbine_type
```

reegis.feedin.**feedin_pv_sets**(*weather*, *location*, *pv_parameter_set*)
Create a pv feed-in time series from a given weather data set and a set of pvlib parameter sets. The result of every parameter set will be a column in the resulting DataFrame.

Parameters

- **weather** (*pandas.DataFrame*) – Weather data set. See module header.
- **location** (*pvlib.location.Location*) – Location of the weather data.
- **pv_parameter_set** (*dict*) – Parameter sets can be created using *create_pvlib_sets()*.

Returns

Return type pandas.DataFrame

reegis.feedin.**feedin_pvlib**(*location*, *system*, *weather*, *tilt=None*, *peak=None*, *orientation_strategy=None*, *installed_capacity=1*)
Create a pv feed-in time series from a given weather data set and a valid pvlib parameter set.

Parameters

- **location** (*pvlib.location.Location or dict*) – Location of the weather data.
- **system** (*dict*) – System parameter for the pvlib.
- **weather** (*pandas.DataFrame*) – Weather data set. See file header for more information.
- **tilt** (*float*) – The tilt angle of the surface. This value can also be defined directly in the system dictionary..
- **peak** (*float*) – Peak power of the pv-module. This value can also be defined directly in the system dictionary.
- **orientation_strategy** (*str*) – See the pvlib documentation for different strategies.
- **installed_capacity** (*float*) – Overall installed capacity for the given pv module. The installed capacity is set to 1 by default for normalised time series.

reegis.feedin.**feedin_wind_sets**(*weather*, *wind_parameter_set*)

Create a wind feed-in time series from a given weather data set and a set of wind parameter sets. The result of every parameter set will be a column in the resulting DataFrame.

Parameters

- **weather** (*pandas.DataFrame*) – Weather data set. See module header.
- **wind_parameter_set** (*dict*) – Parameter sets can be created using *create_windpowerlib_sets()*.

Returns

Return type pandas.DataFrame

Examples

```
>>> from reegis import coastdat
>>> fn=os.path.join(os.path.dirname(__file__), os.pardir, 'tests',
...                 'data', 'test_coastdat_weather.csv')
>>> wind_parameter_set=create_windpowerlib_sets()
>>> weather=pd.read_csv(fn, header=[0, 1])['1126088']
>>> data_height=cfg.get_dict('coastdat_data_height')
>>> wind_weather=coastdat.adapt_coastdat_weather_to_windpowerlib(
...     weather, data_height)  # doctest: +SKIP
>>> feedin_wind_sets(wind_weather, wind_parameter_set
...     ).sum().sort_index()  # doctest: +SKIP
ENERCON_82_hub138_2300    1673.216046
ENERCON_82_hub78_3000     1048.678195
ENERCON_82_hub98_2300     1487.604336
dtype: float64
```

reegis.feedin.**feedin_windpowerlib**(*weather*, *turbine*, *installed_capacity=1*)

Use the windpowerlib to generate normalised feedin time series.

Parameters

- **turbine** (*dict or windpowerlib.wind_turbine.WindTurbine*) – Parameters of the wind turbine (hub height, diameter of the rotor, identifier of the turbine to get cp-series, nominal power).

- **weather** (*pandas.DataFrame*) – Weather data set. See module header.
- **installed_capacity** (*float*) – Overall installed capacity for the given wind turbine. The installed capacity is set to 1 by default for normalised time series.

Returns

Return type pandas.DataFrame

Examples

```
>>> from reegis import coastdat
>>> fn=os.path.join(os.path.dirname(__file__), os.pardir, 'tests',
...                 'data', 'test_coastdat_weather.csv')
>>> weather=pd.read_csv(fn, header=[0, 1])['1126088']
>>> turbine={
...     'hub_height': 135,
...     'rotor_diameter': 127,
...     'name': 'E-82/2300',
...     'nominal_power': 4200000,
...     'fetch_curve': 'power_coefficient_curve'}
>>> data_height=cfg.get_dict('coastdat_data_height')
>>> wind_weather=coastdat.adapt_coastdat_weather_to_windpowerlib(
...     weather, data_height)  # doctest: +SKIP
>>> int(feedin_windpowerlib(wind_weather, turbine).sum())  # doctest: +SKIP
1737
```

reegis.feedin.**get_optimal_pv_angle**(*lat*)

About 27° to 34° from ground in Germany. The pvlib uses tilt angles horizontal=90° and up=0°. Therefore 90° minus the angle from the horizontal.

10.5 reegis.geometries module

Reegis geometry tools.

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.geometries.**create_geo_df**(*df*, *wkt_column=None*, *lon_column=None*, *lat_column=None*, *crs=None*)

Convert pandas.DataFrame to geopandas.geoDataFrame

reegis.geometries.**get_federal_states_polygon**()

Get a region set for the federal states of Germany.

Examples

```
>>> list(get_federal_states_polygon().iloc[0:4].index)
['HH', 'NI', 'MV', 'SH']
```

reegis.geometries.**get_germany_polygon**(*with_awz=False*)

Get the polygon of Germany with the exclusive economic zone of Germany in one polygon.

Examples

```
>>> int(get_germany_polygon(with_awz=True).to_crs(epsg=25832).area[0]/1e6)
414537
>>> int(get_germany_polygon(with_awz=False).to_crs(epsg=25832).area[0]/1e6)
357047
```

reegis.geometries.**get_germany_with_awz_polygon**()

reegis.geometries.**lat_lon2point**(*df*)
Create shapely point object of latitude and longitude.

reegis.geometries.**load**(*path=None*, *filename=None*, *fullname=None*, *hdf_key=None*, *index_col=None*, *crs=None*)
Load files with geographic information into a GeoDataFrame.

Allowed types are csv, hdf, shp and geojson.

reegis.geometries.**load_csv**(*path=None*, *filename=None*, *fullname=None*, *index_col=None*)
Load csv-file into a DataFrame.

reegis.geometries.**load_hdf**(*path=None*, *filename=None*, *fullname=None*, *key=None*)
Load a *hdf* file.

reegis.geometries.**load_shp**(*path=None*, *filename=None*, *fullname=None*)
Load an *shp* file as GeoDataFrame.

reegis.geometries.**remove_invalid_geometries**(*gdf*)
Remove rows that do not have a valid geometry.

reegis.geometries.**spatial_join_with_buffer**(*geo1*, *geo2*, *name*, *jcol='index'*, *step=0.05*, *limit=1*)
Add name of containing region to new column for all points.

Parameters

- **geo1** (*geopandas.geoDataFrame*) – Point layer.
- **geo2** (*geopandas.geoDataFrame*) – Polygon layer.
- **jcol** (*str*) –
- **name** (*str*) – Name of the new column with the region names/identifiers.
- **step** (*float*) –
- **limit** (*float*) –

Returns

Return type geopandas.geoDataFrame

10.6 reegis.inhabitants module

Aggregate the number of inhabitants for a regions/polygons within Germany.

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.inhabitants.**get_ew_by_federal_states**(*year*)
Get the inhabitants per federal state for a given year.

reegis.inhabitants.**get_ew_geometry**(*year*, *polygon=False*)
Get a map with the number of inhabitants.

reegis.inhabitants.**get_ew_shp_file**(*year*)

Parameters **year** –

Examples

```
>>> print(get_ew_shp_file(2014)[-35:])
data/inhabitants/VG250_VWG_2014.shp
```

reegis.inhabitants.**get_inhabitants_by_multi_regions**(*year*, *geo*, *name*)
Get a MultiIndex table with the inhabitants from all given geometry sets.

Parameters

- **year** (*int*) –
- **geo** (*tuple or list*) –
- **name** (*tuple or list*) –

Examples

```
>>> geo1=geometries.load(
...     cfg.get('paths', 'geometry'),
...     cfg.get('geometry', 'de21_polygons'), index_col='region')
>>> geo2=geometries.get_federal_states_polygon()
>>> inh=get_inhabitants_by_multi_regions(
...     2014, [geo1, geo2], ['de21', 'fs'])
>>> inh.loc['DE01']['BB']
1811137
>>> inh.loc['DE01']['BE']
3469849
```

reegis.inhabitants.**get_inhabitants_by_region**(*year*, *geo*, *name*)
Get inhabitants for the given region polygons.

Parameters

- **year** –
- **geo** –
- **name** –

Returns

Return type pd.DataFrame

Examples

```
>>> geo=geometries.get_federal_states_polygon()
>>> get_inhabitants_by_region(2014, geo, name='federal_states').sum()
81197537
```

reegis.inhabitants.**get_share_of_federal_states_by_region**(*year*, *regions*, *name*)

Parameters

- **year** (*int*) –
- **regions** (*tuple or list*) –
- **name** (*tuple or list*) –

Examples

```
>>> regions=geometries.load(
...     cfg.get('paths', 'geometry'),
...     cfg.get('geometry', 'de21_polygons'), index_col='region')
>>> inh=get_share_of_federal_states_by_region(2014, regions, 'de21')
>>> round(inh.loc['DE01']['BB'], 2)
0.74
>>> round(inh.loc['DE01']['BE'], 2)
1.0
```

10.7 reegis.oedb module

Excess the oedb to get demand data..

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.oedb.**oedb**(*oep_url*, *schema*, *table*, *query*, *geo_column*, *epsg*)
Create a geoDataFrame from a oedb selection.

Examples

```
>>> basic_url='http://oep.iks.cs.ovgu.de/api/v0'
>>> my_request={
...     'schema': 'model_draft',
...     'table': 'ego_demand_hv_largescaleconsumer',
...     'geo_column': 'geom_centre',
...     'query': '',  # '?where=version=v0.4.5'
...     'epsg': 3035}
>>> consumer=oedb(basic_url, **my_request)
>>> int(pd.to_numeric(consumer['consumption']).sum())
26181
```

reegis.oedb.**wkb2wkt**(*x*)
Converts the binary postgis format to WKT such as 'POINT (12.5 53.1)'

10.8 reegis.openego module

Processing the openego map for the electricity demand.

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.openego.**download_oedb**(*oep_url*, *schema*, *table*, *query*, *fn*, *overwrite=False*)
Download map from oedb in WGS84 and store as csv file.

reegis.openego.**get_ego_data**(*osf=True*, *sectors=False*, *query='?where=version=v0.4.5'*)

Parameters

- **osf** (*bool*) – If True the file will be downloaded from the osf page instead of of selected from the database. You may not get the latest version. (default: False)
- **sectors** (*bool*) – By default (False) only the total comsumption is returned. If "True" the consumption devided by sectors will be returned.
- **query** (*str*) – Database query to filter the data set. (default: '?where=version=v0.4.5')

Examples

```
>>> from reegis import openego
>>> # download from file (faster)
>>> openego.get_ego_data()  # doctest: +SKIP
>>> # download from oedb database (get latest updates, very slow)
>>> openego.get_ego_data(osf=False)  # doctest: +SKIP
```

reegis.openego.**get_ego_demand**(*filename=None*, *sectors=False*, *overwrite=False*)

Parameters

- **filename** (*str*) –
- **sectors** (*bool*) –
- **overwrite** (*bool*) –

Returns

Return type pandas.DataFrame

reegis.openego.**get_ego_demand_by_region**(*regions*, *name*, *outfile=None*, *infile=None*, *dump=False*, *grouped=False*, *sectors=False*, *overwrite=False*)
Add the region id from a given region set to the openego demand table. This can be used to calculate the demand or the share of each region.

Parameters

- **regions** (*GeoDataFrame*) – A region set.
- **name** (*str*) – The name of the region set will be used as the name of the column in the openego GeoDataFrame and to distinguish result files.
- **outfile** (*str (optional)*) – It is possible to pass a filename (with path) where the results should be stored. Only valid if *dump* is True.
- **infile** (*str (optional)*) – It is possible to use a specific infile (with path) where the openego map is stored.
- **dump** (*bool*) – If dump is True the result will be returned and stored into a file. Otherwise the result is just returned. (default: False)
- **grouped** (*bool*) – If grouped is False the openego table with a region column is returned. Otherwise the map is grouped by the region column and the consumption column is summed up. (default: False)
- **sectors** (*bool*) – Still missing.

- **overwrite** (*bool*) –

Returns **pandas.DataFrame or pandas.Series** – True.

Return type A Series is returned if grouped is

Notes

The openego map may not be updated in the future so it might be necessary to scale the results to an overall demand.

Examples

```
>>> federal_states=geometries.get_federal_states_polygon()
>>> bmwi_annual=bmwi_data.get_annual_electricity_demand_bmwi(
...     2015)  # doctest: +SKIP
```

```
>>> egodemand=get_ego_demand_by_region(
...     federal_states, 'federal_states', grouped=True)  # doctest: +SKIP
```

```
>>> egodemand.div(ego_demand.sum()).mul(bmwi_annual)  # doctest: +SKIP
```

reegis.openego.**wkb2wkt**(*x*)
Loads geometry from wkb.

10.9 reegis.opsd module

10.10 reegis.powerplants module

10.11 reegis.storages module

Processing a list of power plants in Germany.

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.storages.**lat_lon2point**(*df*)
Create shapely point object of latitude and longitude.

reegis.storages.**pumped_hydroelectric_storage_by_region**(*regions*, *year*, *name=None*)
Fetch pumped hydroelectric storage by region. This function is based on static data. Please adapt the source file for years > 2018.

Parameters

- **regions** (*geopandas.geoDataFrame*) –
- **name** (*str or None*) –

Returns

Return type pd.DataFrame

Examples

```
>>> federal_states=geometries.get_federal_states_polygon()
>>> phes=pumped_hydroelectric_storage_by_region(
...     federal_states, 2002, 'federal_states')
>>> int(phes.turbine.sum())
5533
>>> phes=pumped_hydroelectric_storage_by_region(
...     federal_states, 2018, 'federal_states')
>>> int(phes.turbine.sum())
6593
>>> int(phes.energy.sum())
37841
>>> round(phes.loc['BW'].pump_eff, 2)
0.86
```

10.12 reegis.mobility module

Calculate the mobility demand.

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.mobility.**calculate_mobility_energy_use**(*year*)

Parameters **year** –

Examples

```
>>> mobility_balance = get_traffic_fuel_energy(2017)
>>> energy_use = calculate_mobility_energy_use(2017)
>>> p = "Petrol usage [TJ]"
>>> d = "Diesel usage [TJ]"
>>> o = "Overall fuel usage [TJ]"
>>> print(p, "(energy balance):", int(mobility_balance["Ottokraftstoffe"]))
Petrol usage [TJ] (energy balance): 719580
>>> print(p, "(calculated):", int(energy_use["petrol"].sum()))
Petrol usage [TJ] (calculated): 803603
>>> print(d, "(energy balance):",
...     int(mobility_balance["Dieselkraftstoffe"]))
Diesel usage [TJ] (energy balance): 1425424
>>> print(d, "(calculated):", int(energy_use["diesel"].sum()))
Diesel usage [TJ] (calculated): 1636199
>>> print(o, "(energy balance):", int(mobility_balance.sum()))
Overall fuel usage [TJ] (energy balance): 2275143
>>> print(o, "(calculated):", int(energy_use.sum().sum()))
Overall fuel usage [TJ] (calculated): 2439803
```

reegis.mobility.**create_grouped_table_kfz**()
Group the kfz-table by main groups.

reegis.mobility.**create_grouped_table_pkw**()
Extract fuel groups of passenger cars

Examples

```
>>> pkw = create_grouped_table_pkw()
>>> pkw['petrol'].sum()
31031021.0
>>> pkw['diesel'].sum()
15153364.0
```

reegis.mobility.**format_kba_table**(*filename*, *sheet*)

Clean the layout of the table.

The tables are made for human readability and not for automatic processing. Lines with subtotals and format-strings of the column names are removed. A valid MultiIndex is created to make it easier to filter the table by the index.

Parameters

- **filename** (*str*) – Path and name of the excel file.
- **sheet** (*str*) – Name of the sheet of the excel table.

Returns

Return type pandas.DataFrame

reegis.mobility.**get_admin_by_region**(*region*)

Allocate admin keys to the given regions.

Parameters **region** (*geopandas.GeoDataFrame*) –

Returns

Return type pd.DataFrame

reegis.mobility.**get_grouped_kfz_by_region**(*region*)

Get the main vehicle groups by region.

Parameters **region** (*geopandas.GeoDataFrame*) –

Returns

Return type pd.DataFrame

Examples

```
>>> fs = geometries.get_federal_states_polygon()
>>> total = get_grouped_kfz_by_region(fs).sum()
>>> int(total["passenger car"])
47095784
>>> int(total["lorry, > 7500"])
295826
```

reegis.mobility.**get_kba_table**()

Get the "kfz" table for all vehicles and the "pkw" table for more statistics about passenger cars.

Returns

Return type namedtuple

Examples

```
>>> table = get_kba_table()
>>> kfz = table.kfz
>>> print(type(kfz))
<class 'pandas.core.frame.DataFrame'>
```

reegis.mobility.**get_mileage_by_type_and_fuel**(*year=2018*)
Get mileage by type and fuel from mileage table and other sources.

See mobility.ini file for more information.

reegis.mobility.**get_mileage_table**()
Download mileage table from the KBA (Kraftfahrtbundesamt) and store it locally.

reegis.mobility.**get_sheet_from_mileage_table**(*sheet*)
Load given sheet from the mileage file.

reegis.mobility.**get_traffic_fuel_energy**(*year*)

Parameters **year** (*int*) –

Examples

```
>>> fuel_energy = get_traffic_fuel_energy(2017)
>>> int(fuel_energy["Ottokraftstoffe"])
719580
>>> fuel_share = fuel_energy.div(fuel_energy.sum()) * 100
>>> round(fuel_share["Dieselkraftstoffe"], 1)
62.7
```

10.13 reegis.tools module

Code snippets without context.

SPDX-FileCopyrightText: 2016-2021 Uwe Krien <krien@uni-bremen.de>

SPDX-License-Identifier: MIT

reegis.tools.**download_file**(*filename*, *url*, *overwrite=False*)
Check if file exist and download it if necessary.

Parameters

- **filename** (*str*) – Full filename with path.
- **url** (*str*) – Full URL to the file to download.
- **overwrite** (*boolean (default False)*) – If set to True the file will be downloaded even though the file exits.

10.14 reegis.coastdat module (experimental)

10.15 Module contents

CHAPTER 11

Figures

11.1 reegis.dev.figures module

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

L

O

P

R

S

W