opendataworkshop

September 27, 2024

1 Open Data Workshop 2024

1.0.1 Open data resources: modelling (& open data publishing)

–> Last year in Sunyani...

1.0.2 Goal:

Find open data resources - to model an existing power grid - to model a future scenario of a power grid

1.0.3 Question:

- What data do we need?

1.0.4 Tools:

- Python for Power System Analysis (PyPSA)
- PyPSA Earth

... Work in Progress

1.0.5 Preparations

[ ]:
```
# Suppress FutureWarnings and UserWarnings
import warnings
warnings.simplefilter('ignore', category=FutureWarning)
warnings.simplefilter('ignore', category=UserWarning)

# Importing the required libraries
import yaml
import fiona
import geopandas as gpd
import pandas as pd
import powerplantmatching as pm
import atlite
import xarray as xr
from pathlib import Path
from scripts._helpers import (two_2_three_digits_country,␣
 ↪two_digits_2_name_country)
```

1.1 Geographical Data

1.1.1 Specify the Country (or Region)

Countries can be specified with country codes according to **ISO 3166-1 alpha-2**

[ ]:
```
# Load the config.yaml file
with open('config.yaml', 'r') as file:
    config = yaml.safe_load(file)

# Get the country code from the config.yaml file
country_code = config.get('countries', [])[0]

# Define the country manually
country_code = 'GH' # 'GH' for Ghana or 'CO' for Colombia
```

[ ]:
```
# Specify the country using the _helper module
country = two_digits_2_name_country(country_code) # 'Ghana' or 'Colombia'
country_gadm = two_2_three_digits_country(country_code) # 'GHA' or 'COL'
print(country_code, country, country_gadm)
```

1.1.2 Country Shapes and Administrative Borders

Import country shapes from **GADM (Global Administrative Areas)**

[ ]:
```
gadm_filename = f"gadm41_{country_gadm}"
gadm_url = f"https://geodata.ucdavis.edu/gadm/gadm4.1/gpkg/{gadm_filename}.gpkg"

gadm_path = 'data/gadm/'
gadm_inputfile_gpkg = gadm_path + gadm_filename + '.gpkg'
gadm_inputfile_gpkg
```

[ ]:
```
import requests
import shutil

# Create directory if it does not exist
Path(gadm_path).mkdir(parents=True, exist_ok=True)

# Download GADM data if it does not exist
if not Path(gadm_inputfile_gpkg).is_file():
    r = requests.get(gadm_url, stream=True, timeout=300)
    with open (gadm_inputfile_gpkg, 'wb') as f:
        shutil.copyfileobj(r.raw, f)
```

[ ]:
```
# List available administrative layers
list_layers = fiona.listlayers(gadm_inputfile_gpkg)
list_layers
```

[ ]:
```
# Plot GADM shape
country_borders = gpd.read_file(gadm_inputfile_gpkg, layer='ADM_ADM_0')
country_borders.boundary.plot()
```

[ ]:
```
shapefile = gpd.read_file(gadm_inputfile_gpkg, layer='ADM_ADM_1')
shapefile.boundary.plot()
```

1.2 Electricity Network

- PyPSA Earth uses **OpenStreetMap (OSM)** data for the electricity network of any country
  - Generators
  - Transmission lines (and cables)
  - Substations (used as buses)
- For Europe, **ENTSO-E** data are available (used by PyPSA-EUR)
- Check the electricity network on **FLOSM**

[ ]:
```
# Path to raw osm files
osm_path = 'resources/' + country_code + '/osm/raw/'

# Lines
lines_file = 'all_raw_lines.geojson'
lines = gpd.read_file(osm_path+lines_file)

# Substations (buses)
buses_file = 'all_raw_substations.geojson'
buses = gpd.read_file(osm_path+buses_file)

# Generators
generators_file = 'all_raw_generators.geojson'
generators = gpd.read_file(osm_path+generators_file)

# Plot network with buses and power plants
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(8,8))
shapefile.boundary.plot(ax=ax, linewidth=0.2, color='darkgreen',␣
 ↪label='Administrative Boundaries')
lines.plot(ax=ax, color='grey', label='Lines')
buses.plot(ax=ax, markersize=25, color='black', label='Substations')
generators.plot(ax=ax, markersize=100, color='blue', label='Power Plants')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
```

1.2.1 Postprocessing OSM Data

PyPSA Earth uses an **algorithm to postprocess** OpenStreetMap (OSM) data for use in energy system models.

1. **Purpose**:
   - Cleans raw OSM data, which includes extracting power-related infrastructure like substations, power lines, and transformers, for use in energy system modeling.

2. **Input Parameters**:
   - Accepts input OSM data in GeoJSON format from the `data/osm` directory.
   - Filters for specific tags related to energy infrastructure (e.g., `substation`, `power lines`, `transformers`).
3. **Cleaning Data**:
   - Filters out irrelevant data and keeps only the records with necessary tags like substations, transformers, and power lines.
4. **Handling Missing Values**:
   - The script removes or fills missing values in important columns to ensure the data is usable for further modeling.
5. **Transforming Coordinates**:
   - Converts the geographical coordinates to a suitable projection system (e.g., EPSG:3035 for Europe) for accurate spatial analysis.
6. **Saving Clean Data**:
   - The cleaned OSM data is saved in GeoJSON format for use in energy system models, specifically in the PyPSA-Earth project.
   - Output files are stored in a predefined directory, typically `resources`.

[ ]:
```
# Path to clean osm files
osm_path = 'resources/' + country_code + '/osm/clean/'

# Lines
lines_file = 'all_clean_lines.geojson'
lines = gpd.read_file(osm_path+lines_file)

# Substations (buses)
buses_file = 'all_clean_substations.geojson'
buses = gpd.read_file(osm_path+buses_file)

# Generators
generators_file = 'all_clean_generators.geojson'
generators = gpd.read_file(osm_path+generators_file)

# Plot network with buses and power plants
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(8,8))
shapefile.boundary.plot(ax=ax, linewidth=0.2, color='darkgreen',␣
 ↪label='Administrative Boundaries')
lines.plot(ax=ax, color='grey', label='Lines')
buses.plot(ax=ax, markersize=25, color='black', label='Substations')
generators.plot(ax=ax, markersize=100, color='blue', label='Power Plants')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
```

1.2.2 Power Plants

PyPSA Earth uses **Powerplantmatching** to improve power plantdata. This is a tool - designed to match and standardize global power plant data from various sources - combines information about power plants, such as their location, capacity, fuel type, and operational status, into a unified

dataset - helps users create a comprehensive and consistent inventory of power plants, which is essential for energy system modeling, analysis of electricity grids, and understanding the role of renewable and conventional power generation in different regions - commonly used in research and projects related to energy planning, decarbonization, and climate policy.

Resources:

1. **Global Power Plant Database (World Resources Institute)**
   A global, open-source dataset containing information about power plants worldwide.
2. **ENTSO-E Transparency Platform**
   The European Network of Transmission System Operators for Electricity provides data on electricity generation and grid operations in Europe.
3. **Platts WEPP (World Electric Power Plants Database)**
   A commercial database that provides detailed information on power plants around the world (subscription required).
4. **Open Power System Data (OPSD)**
   A platform offering data on European power systems, including power plants and electricity generation.
5. **Global Energy Observatory (GEO)**
   A public platform that provides information on energy infrastructure, including power plants and fuel sources.
6. **U.S. Energy Information Administration (EIA)**
   U.S. government agency that provides detailed energy data, including power plant statistics and operational information.
7. **Bundesnetzagentur (Germany)**
   The Federal Network Agency in Germany provides data on electricity generation, grid management, and power plants.

[ ]:
```
# Load config file
config = pm.get_config('configs/powerplantmatching_config.yaml')

# Select target countries
config['target_countries'] = [country]
config['target_countries']
```

[ ]:
```
# Download and process power plants from different sources
ppl_path = 'data/ppl/' + country_code + '/'
ppl_file = 'powerplants.csv'

# Create directory if it does not exist
Path(ppl_path).mkdir(parents=True, exist_ok=True)

if Path(ppl_path + ppl_file).is_file():
    ppl = pd.read_csv(ppl_path + ppl_file, index_col=0)
else:
```

```
    # include solar and wind
    ppl = pm.powerplants(from_url=False, update=True, config_update=config).
 ↪powerplant.fill_missing_decommissioning_years()
    # drop renewable power plants
    #ppl = pm.powerplants(from_url=False, update=True, config_update=config).
 ↪powerplant.fill_missing_decommissioning_years().query('Fueltype not in␣
 ↪["Solar", "Wind"]')
    ppl.to_csv(ppl_path + ppl_file)

ppl.head()
```

Plot Country Shape with Substations and Power Plants

```
[ ]: import matplotlib.pyplot as plt

# Define Coordinate Reference Systems (CRS)
geo_crs = 4326  # general geographic projection, not used for metric measures.␣
 ↪"EPSG:4326" is the standard used by OSM and google maps
area_crs = 6933 # projection for area measurements only. Possible recommended␣
 ↪values are Global Mollweide "ESRI:54009" (but 54009 is not supported by␣
 ↪atlite, use 6933 instead)
# distance_crs = 3857  # projection for distance measurements only. Possible␣
 ↪recommended values are "EPSG:3857" (used by OSM and Google Maps) ->␣
 ↪currently not used

# Read coordinates from power plants and buses
ppl_geometry = gpd.points_from_xy(ppl['lon'], ppl['lat'])
ppl_capacities =gpd.GeoDataFrame(ppl, geometry=ppl_geometry, crs=geo_crs)
#buses_capacities =gpd.GeoDataFrame(buses, geometry=buses_geometry, crs=geo_crs)

# Plot network with buses and power plants
fig, ax = plt.subplots(figsize=(8,8))
shapefile.boundary.plot(ax=ax, linewidth=0.2, color='darkgreen',␣
 ↪label='Administrative Boundaries')
lines.plot(ax=ax, color='grey', label='Lines')
buses.plot(ax=ax, markersize=25, color='black', label='Substations')
ppl_capacities.plot(ax=ax, markersize=100, color='blue', label='Power plants')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
```

1.2.3 Technology Cost Data

PyPSA maintains its own **technology cost database** (see also the Documentation)

- **Purpose**: Provides open-source data for energy technologies to be used in energy system modeling with PyPSA.
- **Content**: Includes technical and economic parameters for energy technologies like power generation, storage, and transmission.
- **Data Sources**: Compiled from government agencies, research institutions, and industry

reports.

- **Use Case**: Helps in simulating future energy systems and analyzing decarbonization scenarios.
- **Goal**: Offers standardized, transparent datasets for research and policy analysis in energy systems.

Sources:

Description	File/Folder/Function in Script
Danish Energy Agency Technology Database, most technologies	technology_data*.xlsx, energy_transport_data_sheet_dec_2017.xlsx
Vartiaien et. al., utility-scale solar PV	function in script
Fraunhofer ISE Studie, gas pipeline costs	Fraunhofer_ISE_*.csv
Older data collected by the PyPSA team from a variety of sources	costs_PyPSA.csv
The European Technology and Innovation Platform for Photovoltaics, solar rooftop	function in script
Lazard, conventional carriers	function in script
Zappe et. al., fuel cost	function in script
Umweltbundesamt – Entwicklung Kohlendioxid, fuel cost	function in script

[ ]:
```
# Load cost data
year = 2020 # can be 2020, 2025, 2030, 2035, 2040, 2045 or 2050
filename = 'data/technology-data/outputs/costs_' + str(year) + '.csv' # The CSV␣
 ↪file has been created with PyPSA-Earth
filename
costs = pd.read_csv(filename, index_col=0)

# Look for PV technology
costs.loc['solar']
```

1.2.4 Electricity consumption and prediction

PyPSA Earth uses the **Global Energy GIS** to predict demand.

Population and GDP prediction is based on **Shared Socioeconomic Pathways** scenarios, data are available from the **IIASA Website**

[ ]:
```
# TODO: Create a function to calculate the demand profile
# Hourly demand profile per bus
load_profile_file = 'data/demand/' + country_code + '/demand_profiles.csv' #␣
 ↪The CSV file has been created with PyPSA-Earth
load_profile = pd.read_csv(load_profile_file, index_col=0, parse_dates=True)

# The peak load for Colombia is less than the installed hydropower capacity,␣
 ↪let's increase it by 40%
if country_code == 'CO':
    load_factor = 1.4
```

```
else:
    load_factor = 1
load_profile = load_profile * load_factor
load_profile
```

1.2.5 Determining Renewable Potentials

Climate & Renewable Energy Data:

- **Climate Data Store:**
  - If you want to use the API (Application Programming Interface), you need to register: https://cds.climate.copernicus.eu/how-to-api
  - Most useful datasets for Renewable Energy: ERA5-Land hourly data from 1950 to present
- **Photovoltaic Geographical Information System (PVGIS)**: Uses SARAH2 and ERA5 datasets
- **Renwables.ninja**: Uses MERRA2 reanalysis and SARAH2 datasets

PyPSA uses **Atlite** for converting weather and climate data into time series of renewable energy potential. It is widely used in energy system modeling and research.

- **Key Features**:
  - Converts weather data (e.g., wind speeds, solar radiation) into usable data for energy system modeling.
  - Supports various renewable energy sources, including wind, solar, and hydro power.
  - Utilizes spatial and temporal data from global climate datasets, such as ERA5 or SARAH.
  - Provides tools to calculate energy availability based on location, time, and technology parameters.
- **Input Data**: Works with climate reanalysis datasets (e.g., ERA5) and Geographic Information System (GIS) data.
- **Output**: Delivers spatially and temporally resolved energy generation potential, often used in energy system optimization.

[ ]:
```
# Example from the website:
cutout = atlite.Cutout(
    path="western-europe-2011-01.nc",
    module="era5",
    x=slice(-13.6913, 1.7712),
    y=slice(49.9096, 60.8479),
    time="2011-01",
)

#cutout.prepare() # Currently throws an error
```

1.2.6 Plotting Solar and Wind Resources

[ ]:
```
# Load the cutout
cutout_file = 'cutouts/' + country_code + '/cutout-2013-era5.nc'
cutout = atlite.Cutout(cutout_file)
cutout
```

[ ]:
```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs

# Calculate the wind and influx
wnd100m = cutout.data.wnd100m.mean(dim="time")
influx = cutout.data['influx_direct'].mean(dim="time") + cutout.
 ↪data['influx_diffuse'].mean(dim="time")

# Get the total bounds of the shapefile
minx, miny, maxx, maxy = shapefile.total_bounds

# Plot the wind and influx
# TODO: check the units and print them on the legend
import matplotlib.pyplot as plt
import cartopy.crs as ccrs

# Create a figure with two subplots
fig, ax = plt.subplots(1, 2, subplot_kw={'projection': ccrs.PlateCarree()},␣
 ↪figsize=(15, 7))

# Set titles for each subplot
# TODO: titles not visible
ax[0].set_title("Mean Wind Potential in m/s", fontsize=14, pad=20)
ax[0].set_extent([minx, maxx, miny, maxy], crs=ccrs.PlateCarree())
ax[1].set_title("Mean Influx in W/m2", fontsize=14)
ax[1].set_extent([minx, maxx, miny, maxy], crs=ccrs.PlateCarree())

# Plot data on the first subplot
wnd100m.plot(ax=ax[0], cmap="Blues")
shapefile.to_crs(geo_crs).plot(ax=ax[0], edgecolor="k", color="none")

# Plot data on the second subplot
influx.plot(ax=ax[1], cmap="Reds")
shapefile.to_crs(geo_crs).plot(ax=ax[1], edgecolor="k", color="none")

# Adjust layout to prevent overlap
plt.tight_layout(rect=[0, 0, 1, 0.95])

# Adjust the space around the titles if necessary
#plt.subplots_adjust(top=5)  # Adjust the value if needed
```

```
# Display the figure
plt.show()
```

1.2.7 Calculate and Plot Wind and Solar Potential

[ ]:
```
# Calculate the area of each cell in the cutout
area = cutout.grid.to_crs(area_crs).area / 1e6
area = xr.DataArray(area.values.reshape(cutout.shape), [cutout.coords["y"],␣
↪cutout.coords["x"]])

#area
```

[ ]:
```
# Define renewable resources
resources = [
    {'method': 'wind', 'turbine': 'Vestas_V112_3MW', 'capacity_per_sqkm': 4.6,␣
↪'resource': 'onwind'},
    {'method': 'pv', 'panel': 'CSi', 'orientation': 'latitude_optimal',␣
↪'capacity_per_sqkm': 2, 'resource': 'solar'},
]
correction_factor = 1

# Create new geodataframe for the capacities
capacities = shapefile
datasets = {}
shape = shapefile.set_index("NAME_1")

# Calculate capacities per resource
# TODO: check the algorithm for correctness and efficiency
for resource in resources:
    method =  resource['method']
    res = resource['resource']
    profile_path = 'resources/' + country_code + '/renewable_profiles/profile_'␣
↪+ res + '.nc'
    if Path(profile_path).is_file():
        print('Profile found:', res)
        ds = xr.open_dataset(profile_path)
        profile = ds['profile']
        capacity = ds['capacities']
        capacities[method] = capacity
        datasets[res] = ds
    else:
        print('Profile not found:', res)
        cap_per_sqkm = resource['capacity_per_sqkm']
        print(method)
        params = [resource.pop(key) for key in ['method', 'capacity_per_sqkm',␣
↪'resource']][0]
```

```
        print(resource)
        func = getattr(cutout, params)
        capacity_factor = correction_factor * func(capacity_factor=True,␣
 ↪**resource)
        #print(capacity_factor)
        layout = capacity_factor * area * cap_per_sqkm
        #print(layout)
        profile, capacity = func(shapes=shape, per_unit=True,␣
 ↪return_capacity=True, layout=layout, **resource)
        #print(capacity)
        capacities[method] = capacity
        ds = xr.Dataset({
        'profile': profile.rename({'NAME_1': 'bus'}),
        'capacities': capacity.rename({'NAME_1': 'bus'})
        })
        datasets[res] = ds
        ds.to_netcdf(profile_path)

datasets
```

[ ]:
```
capacities['area'] = capacities.to_crs(area_crs).area / 1e6 # convert to km2
capacities['area'] = capacities['area'].round(1)
capacities['wind'] = capacities['wind'].round(1)
capacities['pv'] = capacities['pv'].round(1)
capacities['wind_per_sqkm'] = capacities['wind'] / capacities['area']
capacities['pv_per_sqkm'] = capacities['pv'] / capacities['area']
#capacities
```

[ ]:
```
# Plot wind and solar potential in subplots
# TODO: check units
ncols = 3
nrows = 2
fig, ax = plt.subplots(nrows=nrows, ncols=ncols, subplot_kw={'projection': ccrs.
 ↪PlateCarree()}, figsize=(20, 15))

for i in range(nrows):
    for j in range(ncols):
        ax[i, j].set_extent([minx, maxx, miny, maxy], ccrs.PlateCarree())

capacities.plot(ax=ax[0, 0], column="wind", legend=True, cmap="Blues", )
ax[0,0].set_title("Wind Potential in MW")
capacities.apply(lambda x: ax[0,0].annotate(text=x['NAME_1'], xy=x.geometry.
 ↪centroid.coords[0], fontsize=10, ha='center'), axis=1)
capacities.plot(ax=ax[0, 1], column="wind_per_sqkm", legend=True, cmap="Blues")
capacities.apply(lambda x: ax[0,1].annotate(text=x['NAME_1'], xy=x.geometry.
 ↪centroid.coords[0], fontsize=10, ha='center'), axis=1)
ax[0,1].set_title('Wind Potential in MW per km²')
```

```
wnd100m.plot(ax=ax[0, 2], cmap="Blues")
ax[0,2].set_title("Wind Resource in W/m²")
shapefile.to_crs(geo_crs).plot(ax=ax[0, 2], edgecolor="k", color="none")

# PV
capacities.plot(ax=ax[1, 0], column="pv", legend=True, cmap="Reds")
ax[1,0].set_title("PV Potential in MW")
capacities.apply(lambda x: ax[1,0].annotate(text=x['NAME_1'], xy=x.geometry.
 ↪centroid.coords[0], fontsize=10, ha='center'), axis=1)
capacities.plot(ax=ax[1, 1], column="pv_per_sqkm", legend=True, cmap="Reds")
ax[1,1].set_title('PV Potential in MW per km²')
capacities.apply(lambda x: ax[1,1].annotate(text=x['NAME_1'], xy=x.geometry.
 ↪centroid.coords[0], fontsize=10, ha='center'), axis=1)
influx.plot(ax=ax[1, 2], cmap="Reds")
ax[1,2].set_title("Solar Resource in W/m²")
shapefile.to_crs(geo_crs).plot(ax=ax[1, 2], edgecolor="k", color="none")
plt.show()
```

1.2.8 Further Resources

- **Protected Areas:**
  - Data from World Database on Protected Areas (WDPA)
- **Exclusive Economic Zones (EEZ) and Maritime Boundaries**
  - VLIZ EEZ Boundaries
- **Hydrobasins**
  - Data from Hydrosheds