demand-model-ukcp18

August 22, 2024

In this Jupyter notebook, climate model outputs from UKCP18 are applied to a model of infrastructure electricity load.

A weather-driven model of infrastructure electricity load is taken from **Fallon et al. (2023)** and accompanying dataset **Fallon et al. (2024)**.

The Met Office UK has published its UKCP18 climate projections covering a range of climate change scenarios, with simulations spanning 1900 to 2100 and exploring different future emissions pathways.

- The UKCP18 *Global* product comprises projections run on a 60km resolution latitude-longitude grid. 28 projections incorporate 15 members of the HadGEM3-GC3.05 coupled land and ocean-atmosphere Perturbed Parameter Ensemble (PPE), and 13 simulations from the CMIP5 multi-model ensemble.
- The UKCP18 *Regional* product uses a subset of 12 of the HadGEM3-GC3.05 PPE members, applying downscaling over a regino of Europe. Two of the 15 PPE members were excluded from the Regional product due to data gaps, and one of a pair of PPEs (each implementing closely related models) was also excluded (**Murphy et al., 2018**).

This dataset makes use of the 12 PPE members present in the Regional products, and the corresponding 12 members from the Global producs. A full discussion of the methods and research findings can be found in the accompanying paper, **Fallon et al. (in preparation)**

In this notebook you will:
* Read the temperature datasets
* Calculate climate threshold scenarios
* Bias Adjust climate model data
* Create a demand timeseries (using climate model outputs)

If you benefit from using these reanalysis-driven models of infrastructure electricity demand, please consider adding a citation to our research paper **Fallon et al. (in preparation)**.

1 Jupyter notebook setup and configuration

Import required python libraries

[1]:
```
# Access system functions (used to import local modules)
import sys
from os.path import exists

# Maths
```

```
import numpy as np
from numpy.typing import ArrayLike

# Stats / read CSV files
import pandas as pd
from scipy.stats import linregress

# Multi-dimensional data structures
import xarray as xr

# Plotting
import matplotlib.pyplot as plt
import matplotlib as mpl

# Stats plotting
import seaborn as sns

# Hide Warning Messages
import warnings
warnings.filterwarnings('ignore')
```

Import local python files from `modules` folder:

[2]:
```
sys.path.append("modules")

# global temperature threshold scenarios (organise data into convenient␣
 ↪structures for manipulating within the scenario windows)
from scenarios import calculate_hadcrut_reference, calculate_climate_scenarios,␣
 ↪calculate_reanalysis_detrended, calculate_reanalysis_delta, gen_stoch_series

# perform bias adjustments
from bias import QDMBiasAdjust
```

Better default matplotlib style (adjusts fontsize, plotting colours, etc.):

[3]:
```
plt.style.use('seaborn-v0_8-notebook')
sns.set_style("whitegrid")

colors = sns.color_palette("husl", 12)
mpl.rcParams['axes.prop_cycle'] = mpl.cycler(color=colors)
```

Use vector format plots (ensures high DPI render):

[4]:
```
%config InlineBackend.figure_format = 'svg'
```

Compression to use when writing datasets

[5]:
```
encoding_options = dict(zlib=True, complevel=5)
```

2 Datasets

`CSV` files are read using the `pandas` library.

Timeseries data is stored in `data/` and model coefficients are stored in `models/`.

[6]:
```
# read CSV files
merra2_df = pd.read_csv("temperature/merra2-GB-Land-daily.csv",␣
 ↪index_col="time", parse_dates=["time"])
merra2G_df = pd.read_csv("temperature/merra2-Global-Land-monthly.csv",␣
 ↪index_col="time", parse_dates=["time"])
ukcp18_df = pd.read_csv("temperature/ukcp18-GB-Land-daily.csv",␣
 ↪index_col="time")
ukcp18G_df = pd.read_csv("temperature/ukcp18-Global-LandSea-monthly.csv",␣
 ↪index_col="time")

# set column name for UKCP18
ukcp18_df.columns.name = "ppe"
ukcp18G_df.columns.name = "ppe"

# save indexes for later use
merra2_cal = merra2_df.index
merra2G_cal = merra2G_df.index
ukcp18_cal = ukcp18_df.index
ukcp18G_cal = ukcp18G_df.index
```

2.0.1 Convert to xarray

Among other features, datasets can be configured with new co-ordinates by using `xarray`.

[7]:
```
# convert reanalysis to xarray
merra2 = merra2_df.to_xarray().temperature
merra2G = merra2G_df.to_xarray().temperature

# convert climate models to xarray
# (stacking will treat PPE members as a dimension)
ukcp18 = ukcp18_df.stack().to_xarray()
ukcp18G = ukcp18G_df.stack().to_xarray()
```

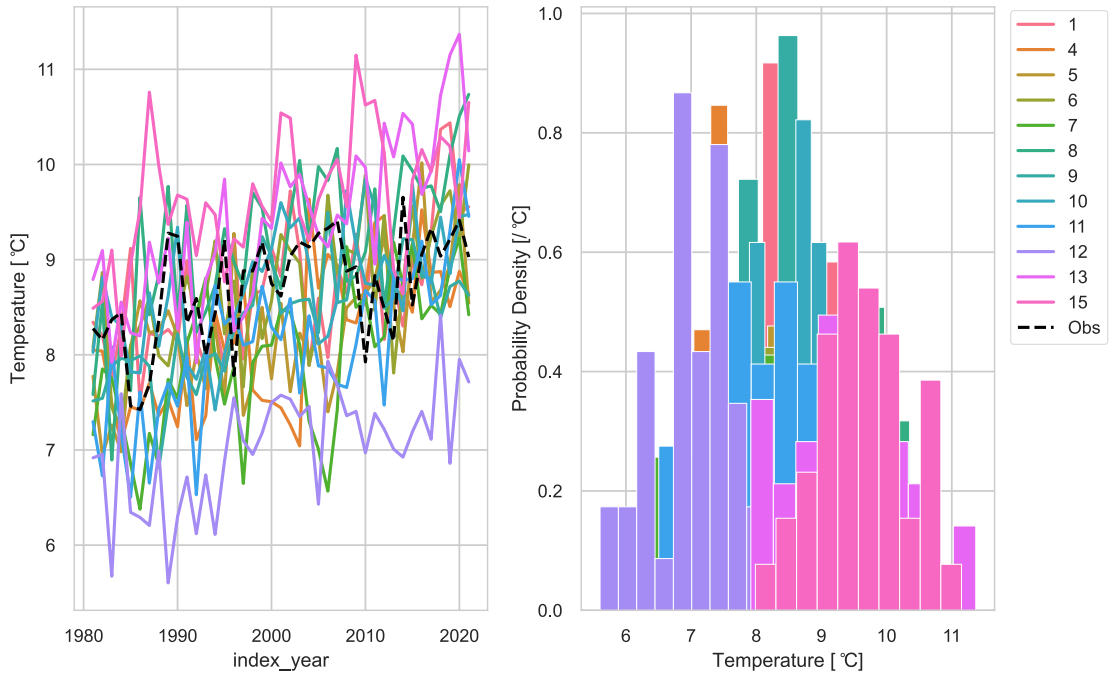
[8]:
```
# create new "index_year" dimension (December to November year)

# merra2 GB (daily)
idx_december = (merra2_cal.month > 11).astype(int)
idx = merra2_cal.year + idx_december
merra2 = merra2.assign_coords(index_year=("time", idx))

# merra2 global (monthly)
idx_december = (merra2G_cal.month > 11).astype(int)
```

```
idx = merra2G_cal.year + idx_december
merra2G = merra2G.assign_coords(index_year=("time", idx))

# ukcp18 GB (daily)
months = np.array(list(int(dt.split("-")[1]) for dt in ukcp18_cal.values))
idx_december = (months > 11).astype(int)
years = np.array(list(int(dt.split("-")[0]) for dt in ukcp18_cal.values))
idx = years + idx_december
ukcp18 = ukcp18.assign_coords(index_year=("time", idx))

# ukcp18 global (monthly)
months = np.array(list(int(dt.split("-")[1]) for dt in ukcp18G_cal.values))
idx_december = (months > 11).astype(int)
years = np.array(list(int(dt.split("-")[0]) for dt in ukcp18G_cal.values))
idx = years + idx_december
ukcp18G = ukcp18G.assign_coords(index_year=("time", idx))
```

2.0.2 Plot GB Temperatures Annual Averages

[9]:
```
# yearly averages
index_year_slice = slice(1981,2021)
merra2T = merra2.groupby("index_year").mean().sel(index_year=index_year_slice)
T = ukcp18.groupby("index_year").mean().sel(index_year=index_year_slice)
```

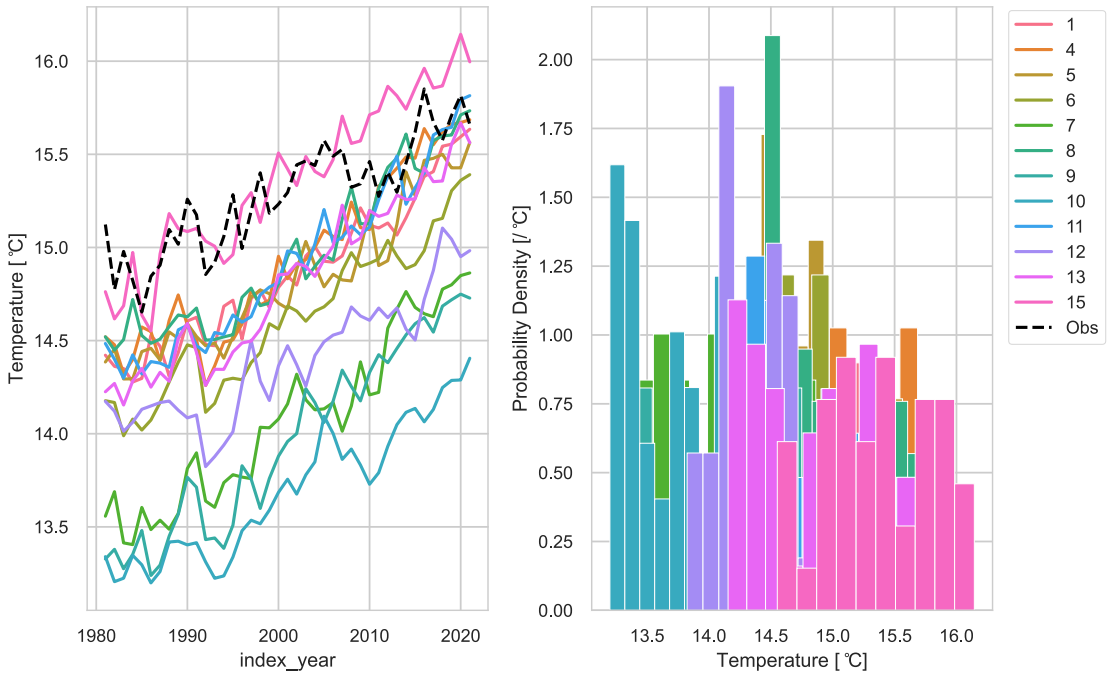
[10]:
```
fig, axs = plt.subplots(1, 2)

for ppe in T.ppe:
    T.sel(ppe=ppe).plot(ax=axs[0], label=int(ppe))
    T.sel(ppe=ppe).plot.hist(ax=axs[1], density=True)
merra2T.plot(ax=axs[0], color="k", ls="--", label="Obs")

axs[0].set_ylabel(r"Temperature [$^\circ\!\!$C]")
axs[1].set_title("")
axs[1].set_xlabel(r"Temperature [$^\circ\!\!$C]")
axs[1].set_ylabel(r"Probability Density [/$^\circ\!\!$C]")
fig.legend(bbox_to_anchor=(1.1,0.98))
fig.tight_layout();
```



2.0.3 Plot Global Temperatures Annual Averages

[11]:
```
merra2TG = merra2G.groupby("index_year").mean().sel(index_year=index_year_slice)
TG = ukcp18G.groupby("index_year").mean().sel(index_year=index_year_slice)
```

[12]:
```
fig, axs = plt.subplots(1, 2)

for ppe in T.ppe:
    TG.sel(ppe=ppe).plot(ax=axs[0], label=int(ppe))
    TG.sel(ppe=ppe).plot.hist(ax=axs[1], density=True)
merra2TG.plot(ax=axs[0], color="k", ls="--", label="Obs")

axs[0].set_ylabel(r"Temperature [$^\circ\!\!$C]")
axs[1].set_title("")
axs[1].set_xlabel(r"Temperature [$^\circ\!\!$C]")
axs[1].set_ylabel(r"Probability Density [/$^\circ\!\!$C]")
fig.legend(bbox_to_anchor=(1.1,0.98))
fig.tight_layout();
```



3 Climate Threshold Scenarios

Using HadCRUT5 as a reference for pre-industrial temperatures, calculate the periods that achieve relative warming of 1, 1.5, 2, 3, 4 °C, as well as the level of warming observed in the reanalysis validation period (1980-2018).

Additionally, for each identified 20-year window, apply a linear-detrending (removing the 20-year trend around the mean value)

[13]:
```
# calculate reference warming from pre-industrial levels to period 1980-2018
hadcrut_ref_period = slice("1980", "2018")
hadcrut_ref = calculate_hadcrut_reference(hadcrut_ref_period)

# define global deltas (present and future warming amounts)
deltas = np.array([hadcrut_ref, 1., 1.5, 2., 3., 4.])
deltas_relative = deltas - deltas[0]
```

[14]:
```
# This code is slow... load from file if possible
# WARNING: If you require a new copy of ukcp18S
#          (eg. because you changed some of the inputs above)
#          then please delete/rename "ukcp18S.nc"
fpath_ukcp18S = "outputs/temperature/ukcp18S.nc"
if not exists(fpath_ukcp18S):
    ukcp18S = calculate_climate_scenarios(merra2, merra2G, ukcp18, ukcp18G,␣
 ↪deltas, hadcrut_ref, hadcrut_ref_period)
```

```
    ukcp18S.encoding.update(encoding_options)
    ukcp18S.to_netcdf(fpath_ukcp18S)
ukcp18S = xr.open_dataarray(fpath_ukcp18S)
```

[15]:
```
ukcp18S
```

[15]:
```
<xarray.DataArray 'temperature' (threshold: 6, ppe: 12, time: 36000)> Size: 21MB
[2592000 values with dtype=float64]
Coordinates:
  * time                      (time) <U10 1MB '1980-12-01' … '2080-11-30'
  * ppe                       (ppe) <U2 96B '1' '4' '5' '6' … '12' '13' '15'
    index_year                (time) int64 288kB …
  * threshold                 (threshold) int64 48B 0 1 2 3 4 5
    threshold_delta           (threshold) float64 48B …
    threshold_delta_relative  (threshold) float64 48B …
Attributes:
    comment:          2m air temperature of GB (from UKCP18 RCM), extracted w…
    history:
    institution:      University of Reading
    licence:          CC BY 4.0
    reference:        https://researchdata.reading.ac.uk
    source:           UKCP18, MERRA-2, HadCRUT5
    title:            UKCP18 Extracted Global Temperature Threshold Scenarios
    production time:  2024-01-31 13:42:31
```

3.1 De-trended reanalysis

We have already calculated global 20-year de-trending for each climate model scenarios (now stored in `outputs/ukcp18S.nc`). Next, calculate the analogous de-trending for the reanalysis data observations.

[16]:
```
# load from file if available
# WARNING: If you require a new copy of merra2_detrended
#          (eg. because you changed some of the inputs above)
#          then please delete/rename "merra2_detrended.nc"
fpath_merra2_detrended = "outputs/temperature/merra2_detrended.nc"
if not exists(fpath_merra2_detrended):
    merra2_detrended = calculate_reanalysis_detrended(merra2, merra2G,␣
 ↪hadcrut_ref_period)
    merra2_detrended.encoding.update(encoding_options)
    merra2_detrended.to_netcdf(fpath_merra2_detrended)
merra2_detrended = xr.open_dataarray(fpath_merra2_detrended)
```

[17]:
```
merra2_detrended
```

[17]:
```
<xarray.DataArray 'temperature' (time: 16072)> Size: 129kB
[16072 values with dtype=float64]
Coordinates:
```

```
  * time         (time) datetime64[ns] 129kB 1980-01-01 1980-01-02 … 2024-01-01
    index_year   (time) int64 129kB …
Attributes:
    comment:           2m air temperature of GB (from MERRA-2 reanalysis)
    history:
    institution:       University of Reading
    licence:           CC BY 4.0
    reference:         https://researchdata.reading.ac.uk
    source:            MERRA-2
    title:             MERRA-2 at Global Temperature Threshold Baseline Scenario
    production time:   2024-01-31 13:56:13
```

4 Bias Adjustment

Using observations (`x`) and climate model historic period outputs (`y`), calculate the bias adjustment to apply to climate model scenarios `z`. See `help(QDMBiasAdjust)` for more information about this operation.

Perform this calculation separately for each combination of `threshold` scenario and `ppe` member (6×12 calculations).

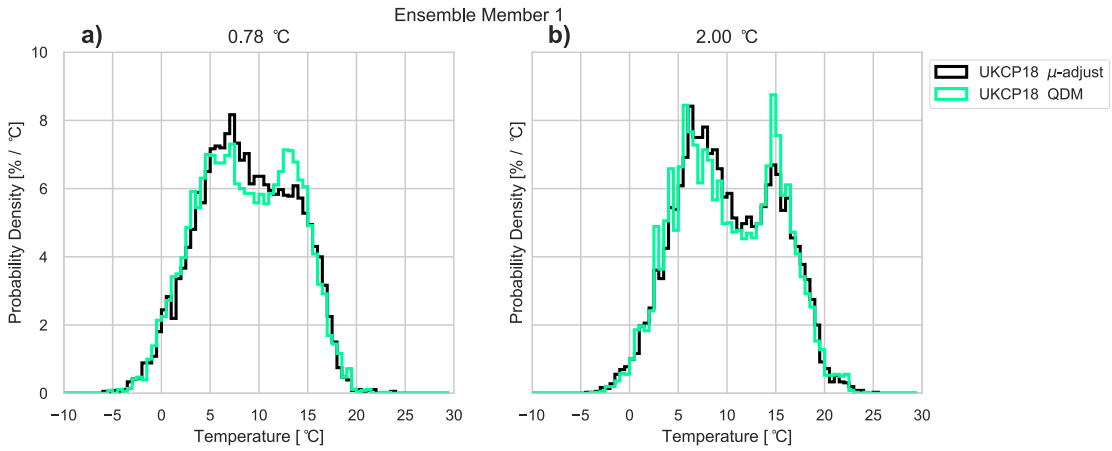
Other bias adjustments from the `bias_adjust.py` sub-module can be substituted in place of Quantile Delta Mapping class, `QDMBiasAdjust`. For example try using as `QMBiasAdjust` or `VarBiasAdjust`.

[19]:
```
# This code takes time... load from file if possible
# WARNING: If you require a new copy of ukcp18S_QDM
#          (eg. because you changed some of the inputs above)
#          then please delete/rename "ukcp18S_QDM.nc"
# initalise array to store bias adjusted values
fpath_ukcp18S_QDM = "outputs/temperature/ukcp18S_QDM.nc"
if not exists(fpath_ukcp18S_QDM):
    ukcp18S_QDM = ukcp18S.copy() + np.nan

    print("Calculating bias-adjustments of PPE... ")
    for iens, ens in enumerate(ukcp18S.ppe):
        print(int(ens), end=", ")

        # calculate quantile map of y (raw) relative to x (obs)
        x = merra2_detrended.squeeze()
        y = ukcp18S.sel(threshold=0, ppe=ens)
        z = ukcp18S.sel(ppe=ens)
        time = list(y.time.values)

        # remove nans from y (will later do this on z, after selecting a given␣
 ↪threshold)
        y = y[~np.isnan(y.values)]
```

```
        # calculate quantile delta map
        QDM = QDMBiasAdjust(x, y, smoothing=True)

        for threshold in z.threshold:
            # get relevant data and drop nans
            w = z.sel(threshold=threshold)
            w = w[~np.isnan(w.values)]

            # apply bias adjustment
            w_corrected = QDM.correct(w)

            # insert bias adjusted values to correct index position
            idxs = (int(threshold), int(iens), slice(time.index(w.time[0]),␣
 ↪time.index(w.time[-1])+1))
            ukcp18S_QDM[idxs] = w_corrected

    ukcp18S_QDM.encoding.update(encoding_options)
    ukcp18S_QDM.to_netcdf(fpath_ukcp18S_QDM)

ukcp18S_QDM = xr.open_dataarray(fpath_ukcp18S_QDM)
```

[20]:
```
fig, axis = plt.subplots(1, 2, sharex=True, sharey=True, figsize=(10, 4))

# choose a PPE member
ensemble_member = "1"

# configure historgram
w = 0.5
bins = np.arange(-10, 30, w)
kwargs = dict(bins=bins, histtype="step", lw=2, density=True)

for ax, panel_label, delta_id in zip(axis, ["a)", "b)"], [0, 3]):
    ukcp18S.sel(ppe=ensemble_member).isel(threshold=delta_id).plot.hist(ax=ax,␣
 ↪color="k", label=r"UKCP18  $\mu$-adjust", **kwargs)
    ukcp18S_QDM.sel(ppe=ensemble_member).isel(threshold=delta_id).plot.
 ↪hist(ax=ax, color="mediumspringgreen", label="UKCP18  QDM", **kwargs)

    ax.set_ylabel(r"Probability Density [% / $^\circ\!\!$C]")
    ax.set_xlabel(r"Temperature [$^\circ\!\!$C]")
    warming = float(ukcp18S.isel(threshold=delta_id).threshold_delta)
    ax.set_title(f"{warming:.2f} $^\circ\!\!$C")
    ax.text(0.1, 1.08, panel_label, transform=ax.transAxes, fontsize=16,␣
 ↪fontweight='bold', va='top', ha='right')

axis[-1].legend(bbox_to_anchor=(1, 1))

# yticks in [percent]
```

```
ylabels = np.arange(0, 11, 2)
yticks = ylabels/100
plt.yticks(yticks, ylabels)

plt.suptitle(f"Ensemble Member {ensemble_member}")
plt.ylim(0, 0.1)
plt.xlim(-10, 30);
```



5 Infrastructure electricity demand model

Calculate the infrastructure electricity demand separately for each entry in `temperature_datasets`. Note that the data must be stored as an `xarray.DataArray` type, with dimensions of `threshold`, `ppe`, `time`.

[21]:
```
temperature_datasets = {
    "merra2_detrended": merra2_detrended.expand_dims(["threshold", "ppe"]),
    "ukcp18S": ukcp18S,
    "ukcp18S_QDM": ukcp18S_QDM
}
```

For a greater number of stochastic realisations, include additional seed numbers, or change the argument to `realisations_size` in the subsequent code window.

[22]:
```
# random number generator (used in stochastic modelling)
rng_list = pd.DataFrame([
    [971218, 801138, 390839, 482617, 944049, 973665, 629269, 577353, 313631,␣
↪777427],
    [658186, 485285, 456551, 518448, 280658, 464875, 517267, 634510, 545247,␣
↪831571],
    [143594, 424747, 791923, 574372, 803152, 484621, 882029, 335722, 298300,␣
↪531536],
```

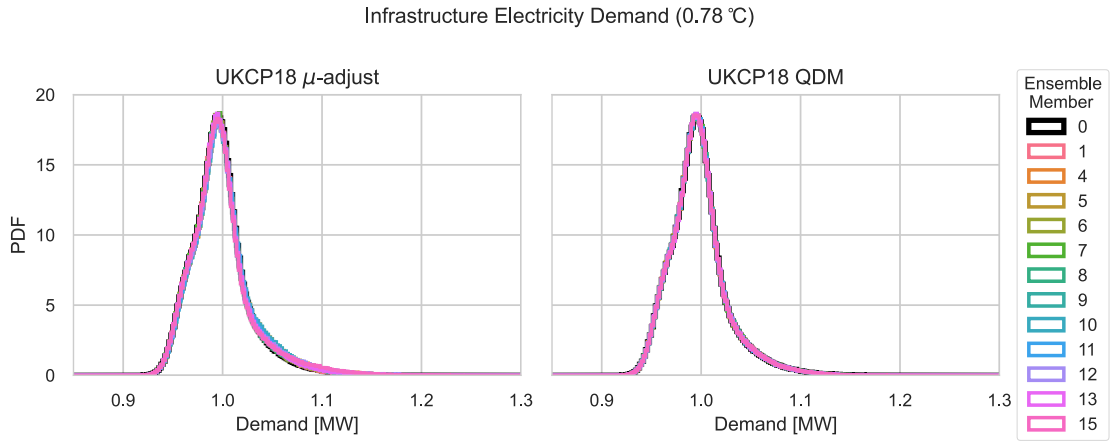
```
], index=temperature_datasets.keys())
```

[23]:
```
# load from file if available
# WARNING: If you require a new copy of a given demand timeseries
#          (eg. because you changed dsome of the inputs above)
#          then please delete/rename the previous .nc file
fpath = "outputs/demand/{name}_{col}.nc"
demand_datasets = {}
for key, dataset in temperature_datasets.items():
    for col, seed in rng_list.loc[key].items():
        rng = np.random.default_rng(seed)
        fname = fpath.format(name=key, col=col)
        if not exists(fname):
            print(f"Generating stochastic time series demand_{key}.nc")
            ds = gen_stoch_series(dataset, realisations_size=70, region="GB",␣
 ↪rng=rng, title=f"{key} derived infrastructure electricity demand")
            ds.encoding.update(encoding_options)
            ds.to_netcdf(fname)
        demand_datasets[key] = xr.open_dataarray(fname)
        break # comment this line to generate all 10 batches of 70 realisations
```

5.1 Plot demand distributions (0.78°C)

[24]:
```
fig, axs = plt.subplots(1, 2, sharex=True, sharey=True, figsize=(9, 3.6))
threshold=0

# histograms
bins = np.arange(0.84, 1.31, 0.002)

# add reanalysis-derived demand to each subplot
for ax in axs:
    demand_datasets["merra2_detrended"].plot.hist(ax=ax, histtype="step", lw=3,␣
 ↪color="k", density=True, bins=bins, label=0)

for ppe in ukcp18S.ppe:
    for ax, key in zip(axs, ["ukcp18S", "ukcp18S_QDM"]):
        demand_datasets[key].sel(threshold=threshold, ppe=ppe).plot.hist(ax=ax,␣
 ↪histtype="step", lw=2, density=True, bins=bins, label=str(ppe.values))

for ax, bias_method in zip(axs, ["UKCP18 $\mu$-adjust", "UKCP18 QDM"]):
    ax.set_xlabel("Demand [MW]")
    ax.set_title(bias_method)

ax.set_ylim(0, 20)
ax.set_yticks(np.arange(0, 21, 5))
ax.set_xlim(0.85, 1.3)
axs[0].set_ylabel("PDF")
```

```
axs[-1].legend(title="Ensemble\n Member", loc="upper left", bbox_to_anchor=(1.
 ↪02,1.12))
fig.suptitle(f"Infrastructure Electricity Demand ({ukcp18S.
 ↪threshold_delta[threshold].values:.2f}" + r"$^\circ\!\!$C)")
fig.tight_layout();
```



5.2 Plot demand distributions (2.0℃)

[25]:
```
fig, axs = plt.subplots(1, 2, sharex=True, sharey=True, figsize=(9, 3.6))
threshold=3

# histograms
bins = np.arange(0.84, 1.31, 0.002)

# add reanalysis-derived demand to each subplot
for ax in axs:
    demand_datasets["merra2_detrended"].plot.hist(ax=ax, histtype="step", lw=3,␣
 ↪color="k", density=True, bins=bins, label=0)

for ppe in ukcp18S.ppe:
    for ax, key in zip(axs, ["ukcp18S", "ukcp18S_QDM"]):
        demand_datasets[key].sel(threshold=threshold, ppe=ppe).plot.hist(ax=ax,␣
 ↪histtype="step", lw=2, density=True, bins=bins, label=str(ppe.values))

for ax, bias_method in zip(axs, ["UKCP18 $\mu$-adjust", "UKCP18 QDM"]):
    ax.set_xlabel("Demand [MW]")
    ax.set_title(bias_method)

ax.set_ylim(0, 20)
ax.set_yticks(np.arange(0, 21, 5))
ax.set_xlim(0.85, 1.3)
```

```
axs[0].set_ylabel("PDF")
axs[-1].legend(title="Ensemble\n Member", loc="upper left", bbox_to_anchor=(1.
 ↪02,1.12))
fig.suptitle(f"Infrastructure Electricity Demand ({ukcp18S.
 ↪threshold_delta[threshold].values:.2f}" + r"$^\circ\!\!$C)")
fig.tight_layout();
```