Urban Climate Explorer

Release v0.0.1

Zhonghua Zheng

Jun 26, 2023

OVERVIEW

Explore and Emulate Urban Climate on AWS Cloud.

Author: Dr. Zhonghua Zheng

CHAPTER

ONE

ABOUT

1.1 Introduction

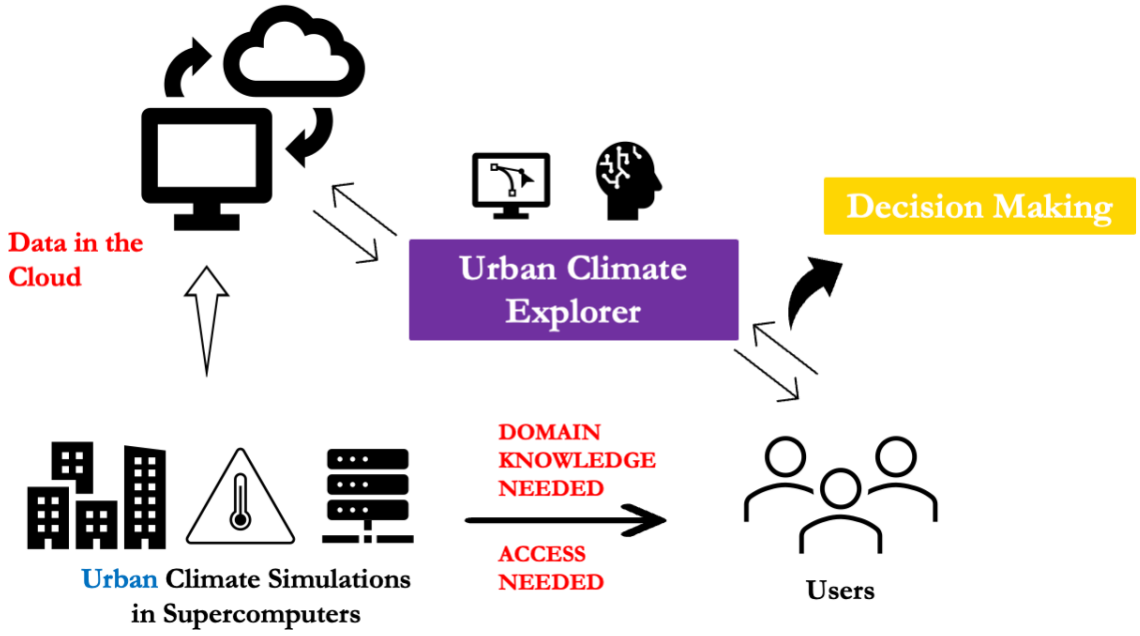
This platform enables free and easy access to the **urban** climate simulations provided by National Center for Atmospheric Research and Amazon Web Services (AWS) via Cloud Computing. By providing the necessary information as the input (e.g., time, latitude, longitude, climate scenarios, etc.), users can explore and utilize urban climate data. Specifically, users can:

- **visualize/analyze** urban climate of a particular city/cities under different climate change scenarios and different version model simulations (e.g., urban heat waves analysis)
- **train** fast machine learning emulators of the urban climate (e.g., mapping from radiation to urban temperature) using a Automated Machine Learning tool (FLAML)
- **apply** the machine learning emulators to users' own data to create customized urban climate projections for their own needs

1.2 Relevant Publications

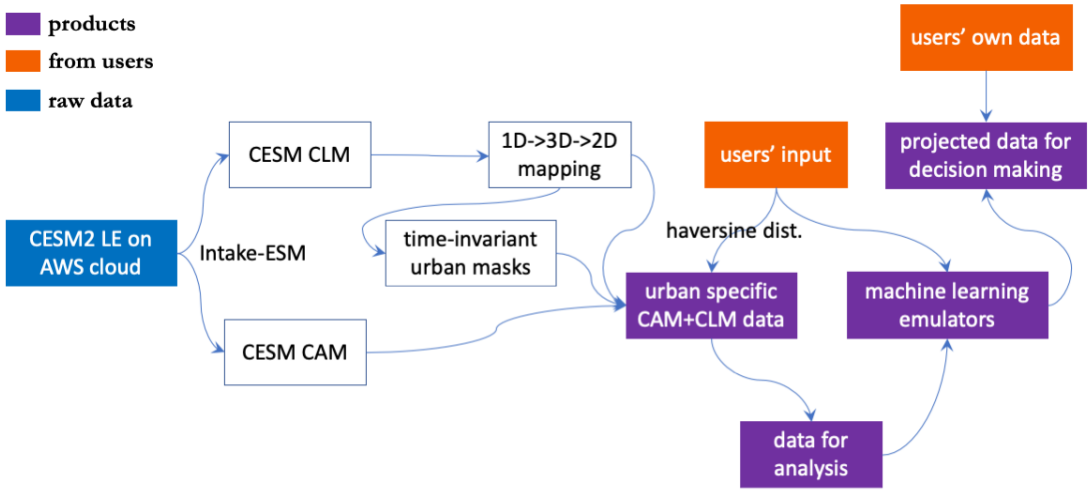
- Zheng, Z., Zhao, L. & Oleson, K.W. Large model structural uncertainty in global projections of urban heat waves. *Nat Commun* **12,** 3736 (2021). https://doi.org/10.1038/s41467-021-24113-9
- Zhao, L., Oleson, K., Bou-Zeid, E. *et al.* Global multi-model projections of local urban climates. *Nat. Clim. Chang.* **11,** 152–157 (2021). https://doi.org/10.1038/s41558-020-00958-8

1.3 Concept



concept

1.4 Technical Workflow



workflow

CHAPTER

TWO

INSTALL AND RUN

2.1 Install

- use conda to install the environment

```
$ git clone git@github.com:zzheng93/UrbanClimateExplorer.git
$ cd ./UrbanClimateExplorer/binder
$ conda env create -f environment.yml
$ conda activate aws_urban
```

2.2 Run

- Locally

```
$ cd ./UrbanClimateExplorer
$ git pull
$ cd ./docs/notebooks
$ jupyter notebook
```

- HPC (e.g., NCAR's Casper clusters with a GPU)
  - First, create a bash script (see below), and name it as `aws_urban_env.sh`, put it in the same folder with your `UrbanClimateExplorer` folder.

```
#!/bin/bash
source  /glade/work/zhonghua/miniconda3/bin/activate aws_urban
echo "ssh -N -L 8889:`hostname`:8889 $USER@`hostname`.ucar.edu"
jupyter notebook --no-browser --ip=`hostname` --port=8889
```

  - Second, run the commands below

    Note: please use `your own job code` instead of "UIUC0021". You can find more information about `execcasper` here

```
$ execcasper -A UIUC0021 -l gpu_type=v100 -l walltime=06:00:00 -l select=1:
↪ncpus=18:mpiprocs=36:ngpus=1:mem=100GB
$ bash aws_urban_env.sh
```

  - Thrid, launch a new terminal, copy and paste the command printed by the "echo" command, and log in. Then open your browser (e.g., Google Chrome), type `https://localhost:8889`.

Note: Sometimes port 8889 may be used by others. In this case, please adjust your bash script accordingly, e.g., from 8889 to 8892:

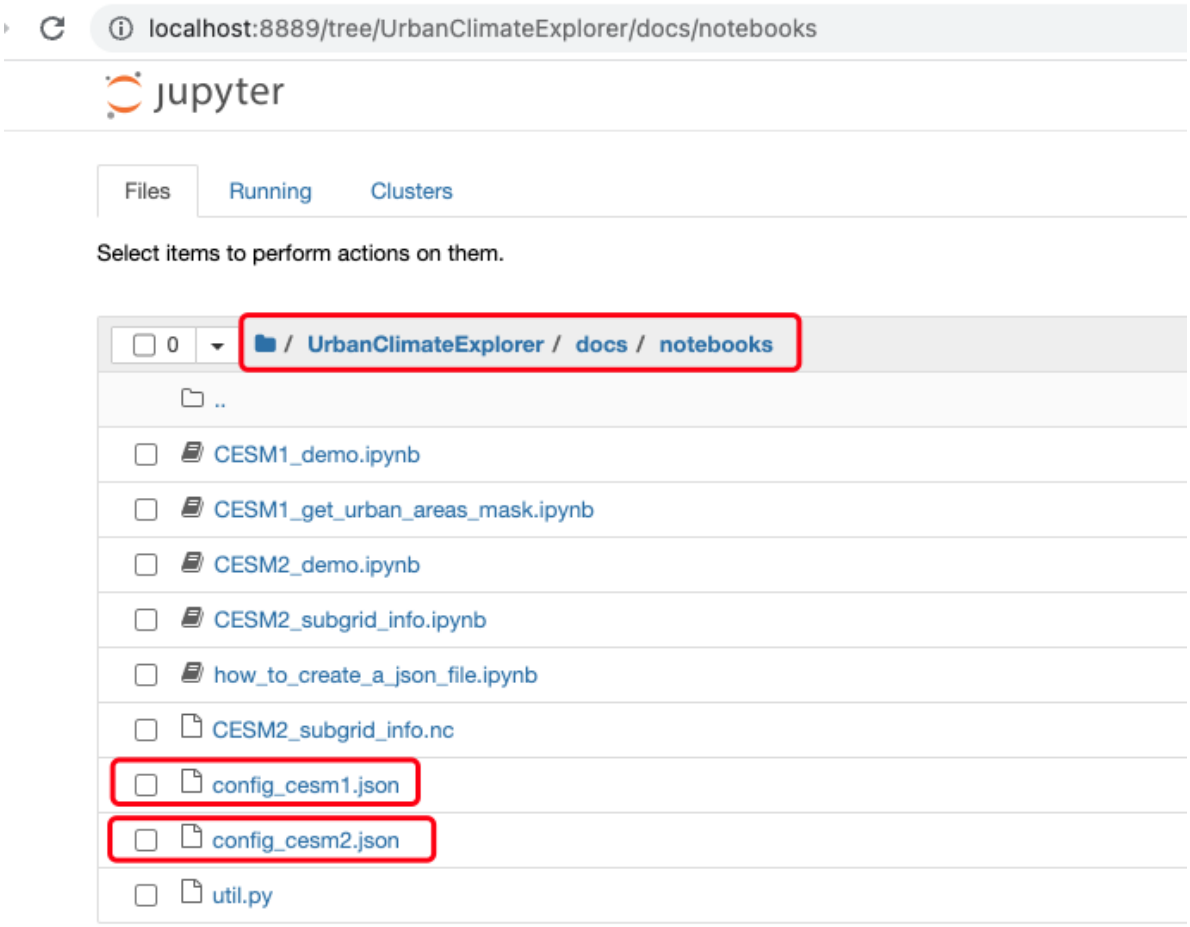
```
#!/bin/bash
source  /glade/work/zhonghua/miniconda3/bin/activate aws_urban
echo "ssh -N -L 8889:`hostname`:8892 $USER@`hostname`.ucar.edu"
jupyter notebook --no-browser --ip=`hostname` --port=8892
```

CHAPTER

THREE

SETUP

- Step 1: Follow Install and Run to launch a jupyter notebook locally or using HPC.
- Step 2: Open `config_cesm1.json` or `config_cesm2.json`.

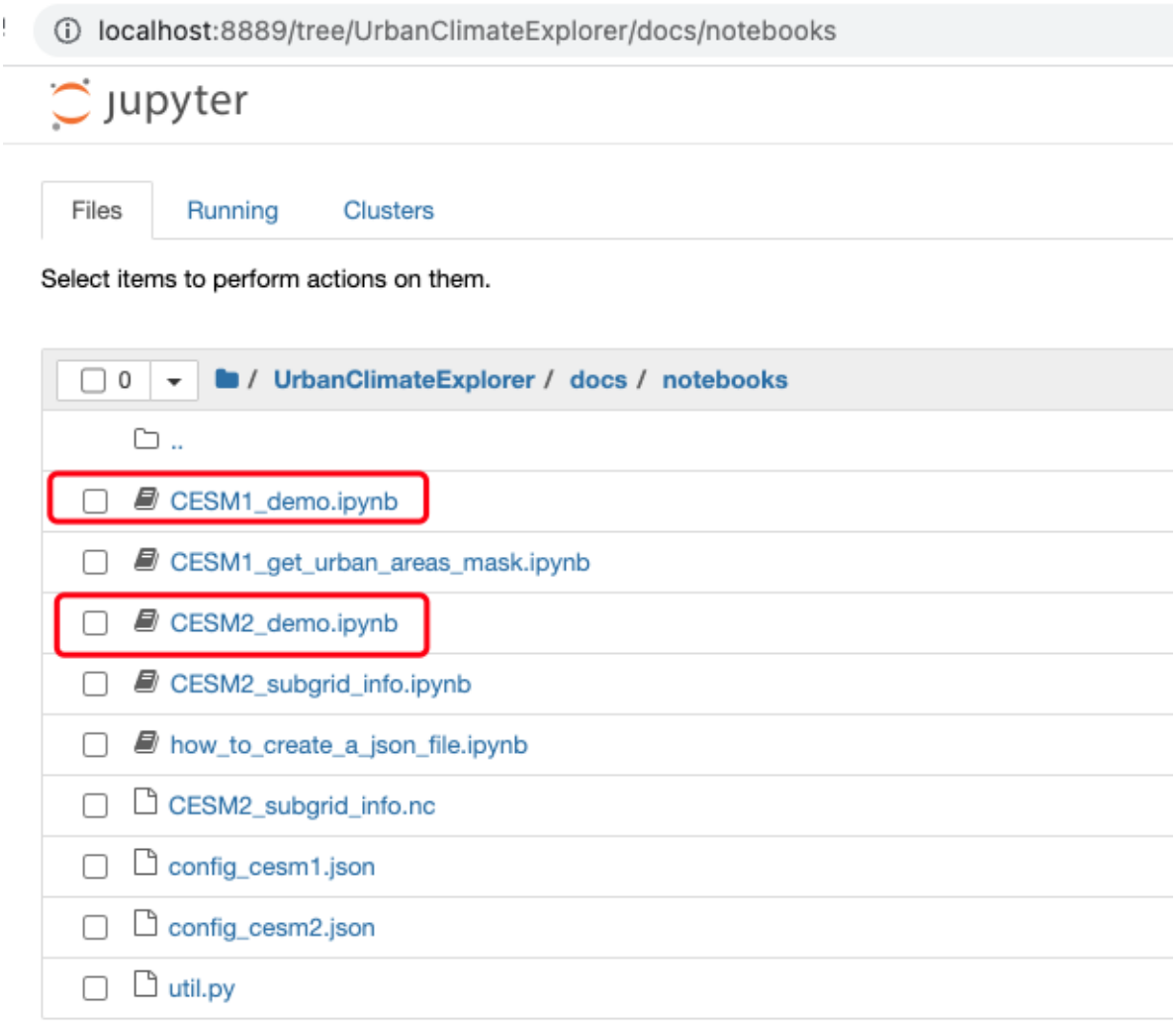


load_JSON

- Step 3: If you are not familiar with CESM1 or CESM2, you can just edit `city_loc` and `time_start` and `time_end`.
  - Please check here for more information about JSON file.
  - Please check here for CESM1 variables and here for CESM2 variables
- Step 4: Save the JSON file

save_JSON

- Step 5: Open `CESM1_demo.ipynb` or `CESM2_demo.ipynb`, depends on which JSON you have edited. Now you are all set!



load_notebook

CHAPTER

FOUR

EXAMPLE FOR CESM1

Reference:
- GitHub: https://github.com/ncar/cesm-lens-aws/
- Data/Variables Information: https://ncar.github.io/cesm-lens-aws/#data-catalog
- Reproduce CESM-LENS: http://gallery.pangeo.io/repos/NCAR/cesm-lens-aws/notebooks/kay-et-al-2015.v3.html

Step 0: load necessary packages and define parameters (no need to change)

[1]:
```
%%time
# Display output of plots directly in Notebook
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import json
from flaml import AutoML
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")
import util

with open("./config_cesm1.json",'r') as load_f:
#     param = json.loads(json.load(load_f))
    param = json.load(load_f)

    model = param["model"] # cesm1
    city_loc = param["city_loc"] # {"lat": 40.0150, "lon": -105.2705}
    l_component = param["l_component"]
    a_component = param["a_component"]
    experiment = param["experiment"]
    frequency = param["frequency"]
    cam_ls = param["cam_ls"]
    clm_ls = param["clm_ls"]
    time = slice(param["time_start"],param["time_end"])
    member_id = param["member_id"]
    estimator_list = param["estimator_list"]
    time_budget = param["time_budget"]
    features = param["features"]
    label = param["label"]
    clm_var_mask = param["label"][0]
```

(continues on next page)

(continued from previous page)

```
# get a dataset
ds = util.get_data(model, city_loc, experiment, frequency, member_id, time, cam_ls, clm_
↪ls)

# create a dataframe
ds['time'] = ds.indexes['time'].to_datetimeindex()
df = ds.to_dataframe().reset_index().dropna()

if "PRSN" in features:
    df["PRSN"] = df["PRECSC"] + df["PRECSL"]

# setup for automl
automl = AutoML()
automl_settings = {
    "time_budget": time_budget,  # in seconds
    "metric": 'r2',
    "task": 'regression',
    "estimator_list":estimator_list,
}
```

```
/glade/work/zhonghua/miniconda3/envs/aws_urban/lib/python3.8/site-packages/xgboost/
↪compat.py:31: FutureWarning: pandas.Int64Index is deprecated and will be removed from␣
↪pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

```
--> The keys in the returned dictionary of datasets are constructed as follows:
        'component.experiment.frequency'
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
CPU times: user 51.6 s, sys: 35.5 s, total: 1min 27s
Wall time: 59.9 s
```

4.1 Step 1: data analysis

xarray.Dataset

[2]:
```
ds
```

[2]:
```
<xarray.Dataset>
Dimensions:     (member_id: 1, time: 7299)
Coordinates:
  * member_id   (member_id) int64 2
    lat         float64 40.05
    lon         float64 255.0
  * time        (time) datetime64[ns] 2081-01-02T12:00:00 ... 2100-12-31T12:0...
Data variables:
    TREFHT      (member_id, time) float32 255.8 266.8 271.1 ... 277.6 276.1
    TREFHTMX    (member_id, time) float32 269.3 278.4 281.2 ... 283.9 277.5
    FLNS        (member_id, time) float32 77.09 67.4 77.49 ... 64.89 37.45 38.35
```

(continues on next page)

(continued from previous page)

```
    FSNS        (member_id, time) float32 83.31 88.83 90.25 ... 67.98 80.27
    PRECSC      (member_id, time) float32 0.0 0.0 0.0 0.0 ... 0.0 0.0 8.927e-12
    PRECSL      (member_id, time) float32 4.887e-10 6.665e-10 ... 9.722e-10
    PRECT       (member_id, time) float32 4.887e-10 6.665e-10 ... 2.32e-08
    QBOT        (member_id, time) float32 0.000913 0.001889 ... 0.005432 0.00492
    UBOT        (member_id, time) float32 5.461 4.815 4.506 ... 2.865 3.255
    VBOT        (member_id, time) float32 1.27 3.189 3.691 ... 1.215 0.8704
    TREFMXAV_U  (member_id, time) float32 259.6 270.0 279.8 ... 284.9 285.3
Attributes: (12/14)
    intake_esm_varname:        FLNS\nFSNS\nPRECSC\nPRECSL\nPRECT\nQBOT\nTREFH...
    topography_file:           /scratch/p/pjk/mudryk/cesm1_1_2_LENS/inputdata...
    title:                     UNSET
    Version:                   $Name$
    NCO:                       4.4.2
    host:                      tcs-f02n07
    ...                        ...
    important_note:            This data is part of the project 'Blind Evalua...
    initial_file:              b.e11.B20TRC5CNBDRD.f09_g16.105.cam.i.2006-01-...
    source:                    CAM
    revision_Id:               $Id$
    logname:                   mudryk
    intake_esm_dataset_key:    atm.RCP85.daily
```

pandas dataframe

[3]:
```
df.head()
```

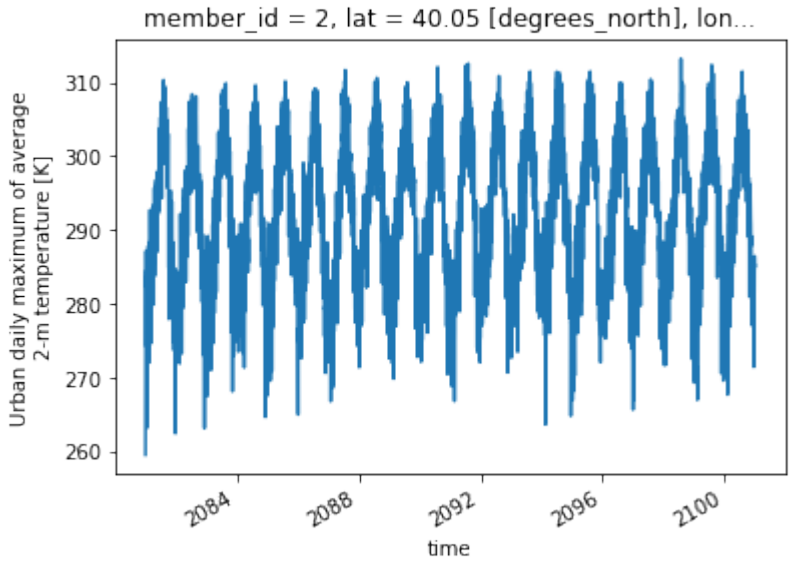
[3]:

	member_id	time	TREFHT	TREFHTMX	FLNS	FSNS	PRECSC	PRECSL	PRECT	QBOT	UBOT	VBOT	lat	lon	TREFMXAV_U	PRSN
0	2	2081-01-02 12:00:00	255.849716	269.327850	77.085800	83.311066	0.0	4.886532e-10	4.886532e-10	0.000913	5.461293	1.270236	40.052357	255.0	259.641968	4.886532e-10
1	2	2081-01-03 12:00:00	266.790833	278.396545	67.402657	88.831192	0.0	6.664702e-10	6.665211e-10	0.001889	4.814545	3.189500	40.052357	255.0	269.972809	6.664702e-10
2	2	2081-01-04 12:00:00	271.122040	281.181946	77.493195	90.247482	0.0	8.030515e-14	2.538048e-11	0.003331	4.506459	3.690698	40.052357	255.0	279.828827	8.030515e-14
3	2	2081-01-05 12:00:00	276.329895	281.844543	64.789749	93.343193	0.0	1.896001e-20	2.508245e-09	0.004209	5.162941	4.963157	40.052357	255.0	282.674103	1.896001e-20
4	2	2081-01-06 12:00:00	275.347229	280.980469	69.647041	80.993706	0.0	5.442079e-17	7.844814e-09	0.004014	4.478484	4.272586	40.052357	255.0	283.518707	5.442079e-17

data visualization

[4]:
```
ds["TREFMXAV_U"].plot()
```

[4]:
```
[<matplotlib.lines.Line2D at 0x2b3bfa600a00>]
```



4.2 Step 2: automated machine learning

train a model (emulator)

[5]:
```
%%time
# assume that we want to split the data into training data and testing data
# let's use first 95% for training, and the remaining for testing
idx = df.shape[0]
train = df.iloc[:int(0.95*idx),:]
test = df.iloc[int(0.95*idx):,:]
(X_train, y_train) = (train[features], train[label].values)
(X_test, y_test) = (test[features], test[label].values)

# train the model
automl.fit(X_train=X_train, y_train=y_train,
           **automl_settings, verbose=-1)
print(automl.model.estimator)
```

```
XGBRegressor(base_score=0.5, booster='gbtree',
             colsample_bylevel=0.8981353296453468, colsample_bynode=1,
             colsample_bytree=0.8589079860800738, gamma=0, gpu_id=-1,
             grow_policy='lossguide', importance_type='gain',
             interaction_constraints='', learning_rate=0.05211228610238813,
             max_delta_step=0, max_depth=0, max_leaves=29,
             min_child_weight=45.3846760978798, missing=nan,
             monotone_constraints='()', n_estimators=299, n_jobs=-1,
             num_parallel_tree=1, random_state=0, reg_alpha=0.01917865165509354,
             reg_lambda=2.2296607355174927, scale_pos_weight=1,
             subsample=0.9982731696185565, tree_method='hist',
             use_label_encoder=False, validate_parameters=1, verbosity=0)
CPU times: user 3min 35s, sys: 2.67 s, total: 3min 38s
Wall time: 15.7 s
```

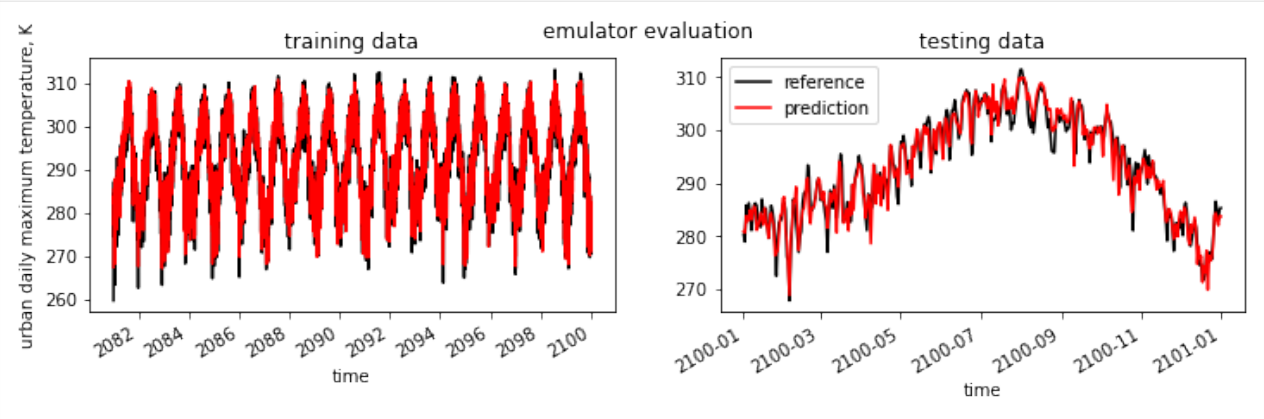
apply and test the machine learning model
use `automl.predict(X)` to apply the model

[6]:
```
# training data
print("model performance using training data:")
y_pred = automl.predict(X_train)
print("root mean square error:",
      mean_squared_error(y_true=y_train, y_pred=y_pred, squared=False))
print("r2:", r2_score(y_true=y_train, y_pred=y_pred),"\n")
import pandas as pd
d_train = {"time":train["time"],"y_train":y_train.reshape(-1),"y_pred":y_pred.reshape(-
↪1)}
df_train = pd.DataFrame(d_train).set_index("time")

# testing data
print("model performance using testing data:")
y_pred = automl.predict(X_test)
print("root mean square error:",
      mean_squared_error(y_true=y_test, y_pred=y_pred, squared=False))
print("r2:", r2_score(y_true=y_test, y_pred=y_pred))
d_test = {"time":test["time"],"y_test":y_test.reshape(-1),"y_pred":y_pred.reshape(-1)}
df_test = pd.DataFrame(d_test).set_index("time")
```

```
model performance using training data:
root mean square error: 1.7566193
r2: 0.9681344385429989

model performance using testing data:
root mean square error: 2.3287773
r2: 0.9384653117109892
```

visualization

[7]:
```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(12,3))
fig.suptitle('emulator evaluation')
df_train["y_train"].plot(label="reference",c="k",ax=ax1)
df_train["y_pred"].plot(label="prediction",c="r",ax=ax1)
ax1.set_title("training data")
ax1.set_ylabel("urban daily maximum temperature, K")

df_test["y_test"].plot(label="reference",c="k",ax=ax2)
df_test["y_pred"].plot(label="prediction",c="r",ax=ax2)
ax2.set_title("testing data")
plt.legend()
plt.show()
```


emulator evaluation
training data
testing data
urban daily maximum temperature, K
310
300
290
280
270
260
2082
2084
2086
2088
2090
2092
2094
2096
2098
2100
time
reference
prediction
2100-01
2100-03
2100-05
2100-07
2100-09
2100-11
2101-01
time

CHAPTER

FIVE

EXAMPLE FOR CESM2

NOTE: Compared to the CESM1 demo, here "Q" (QBOT), "U" (UBOT) and "V" (VBOT) are not included. When the bottom "lev" of "Q", "U", and "V" are merged, there is an issue.

Reference:
- GitHub: https://github.com/NCAR/cesm2-le-aws
- Data/Variables Information: https://ncar.github.io/cesm2-le-aws/model_documentation.html#data-catalog
- Reproduce CESM-LENS: https://github.com/NCAR/cesm2-le-aws/blob/main/notebooks/kay_et_al_lens2.ipynb

Step 0: load necessary packages and define parameters (no need to change)

[1]:
```
%%time
# Display output of plots directly in Notebook
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import json
from flaml import AutoML
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")
import util

with open("./config_cesm2.json",'r') as load_f:
#     param = json.loads(json.load(load_f))
    param = json.load(load_f)

    model = param["model"] # cesm2
    urban_type = param["urban_type"] # md
    city_loc = param["city_loc"] # {"lat": 40.0150, "lon": -105.2705}
    l_component = param["l_component"]
    a_component = param["a_component"]
    experiment = param["experiment"]
    frequency = param["frequency"]
    cam_ls = param["cam_ls"]
    clm_ls = param["clm_ls"]
    forcing_variant = param["forcing_variant"]
    time = slice(param["time_start"],param["time_end"])
    member_id = param["member_id"]
```

(continues on next page)

(continued from previous page)

```
    estimator_list = param["estimator_list"]
    time_budget = param["time_budget"]
    features = param["features"]
    label = param["label"]
    clm_var_mask = param["label"][0]

# get a dataset
ds = util.get_data(model, city_loc, experiment, frequency, member_id, time, cam_ls, clm_
↪ls,
                   forcing_variant=forcing_variant, urban_type=urban_type)

# create a dataframe
ds['time'] = ds.indexes['time'].to_datetimeindex()
df = ds.to_dataframe().reset_index().dropna()

if "PRSN" in features:
    df["PRSN"] = df["PRECSC"] + df["PRECSL"]
if "PRECT" in features:
    df["PRECT"] = df["PRECC"] + df["PRECL"]

# setup for automl
automl = AutoML()
automl_settings = {
    "time_budget": time_budget,  # in seconds
    "metric": 'r2',
    "task": 'regression',
    "estimator_list":estimator_list,
}
```

```
/glade/work/zhonghua/miniconda3/envs/aws_urban/lib/python3.8/site-packages/xgboost/
↪compat.py:31: FutureWarning: pandas.Int64Index is deprecated and will be removed from␣
↪pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

```
--> The keys in the returned dictionary of datasets are constructed as follows:
        'component.experiment.frequency.forcing_variant'
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
different lat between CAM and CLM subgrid info, adjust subgrid info's lat
CPU times: user 56.9 s, sys: 34 s, total: 1min 30s
Wall time: 58.4 s
```

5.1 Step 1: data analysis

xarray.Dataset

[2]:
```
ds
```

[2]:
```
<xarray.Dataset>
Dimensions:    (member_id: 1, time: 7299)
Coordinates:
    lat        float64 40.05
    lon        float64 255.0
  * member_id  (member_id) <U12 'r1i1231p1f1'
  * time       (time) datetime64[ns] 2081-01-02T12:00:00 ... 2100-12-31T12:00:00
Data variables:
    TREFHT     (member_id, time) float32 273.2 273.4 275.7 ... 276.7 277.0 277.4
    TREFHTMX   (member_id, time) float32 276.2 278.8 282.8 ... 283.9 284.8 283.7
    FLNS       (member_id, time) float32 93.1 82.03 82.87 ... 88.49 87.6 67.41
    FSNS       (member_id, time) float32 90.73 84.27 91.37 ... 91.55 91.45 77.9
    PRECSC     (member_id, time) float32 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0
    PRECSL     (member_id, time) float32 4.184e-10 4.287e-10 ... 9.825e-21
    PRECC      (member_id, time) float32 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0
    PRECL      (member_id, time) float32 4.251e-10 4.42e-10 ... 1.181e-08
    TREFMXAV   (member_id, time) float64 277.0 279.3 284.4 ... 284.3 285.7 284.4
Attributes:
    host:                    mom1
    topography_file:         /mnt/lustre/share/CESM/cesm_input/atm/cam/topo/f...
    logname:                 sunseon
    time_period_freq:        day_1
    intake_esm_varname:      FLNS\nFSNS\nPRECC\nPRECL\nPRECSC\nPRECSL\nTREFHT...
    Conventions:             CF-1.0
    model_doi_url:           https://doi.org/10.5065/D67H1H0V
    source:                  CAM
    intake_esm_dataset_key:  atm.ssp370.daily.cmip6
```

pandas dataframe

[3]:
```
df.head()
```

[3]:

	member_id	time	TREFHT	TREFHTMX	FLNS	FSNS	PRECSC	PRECSL	PRECC	PRECL	lat	lon	TREFMXAV	PRSN	PRECT
0	r1i1231p1f1	2081-01-02 12:00:00	273.180023	276.192291	93.097984	90.734787	0.0	4.183626e-10	0.0	4.251142e-10	40.052356	255.0	276.999237	4.183626e-10	4.251142e-10
1	r1i1231p1f1	2081-01-03 12:00:00	273.396667	278.823547	82.032143	84.271416	0.0	4.287326e-10	0.0	4.419851e-10	40.052356	255.0			
2	r1i1231p1f1	2081-01-04 12:00:00	275.675842	282.826111	82.870590	91.365944	0.0	1.492283e-15	0.0	7.843200e-14	40.052356	255.0			
3	r1i1231p1f1	2081-01-05 12:00:00	275.782043	282.312042	90.888451	92.246887	0.0	3.730888e-17	0.0	3.108414e-15	40.052356	255.0			
4	r1i1231p1f1	2081-01-06 12:00:00	272.146301	275.967560	56.035732	55.395706	0.0	5.766720e-11	0.0	9.267757e-11	40.052356	255.0			

(continues on next page)

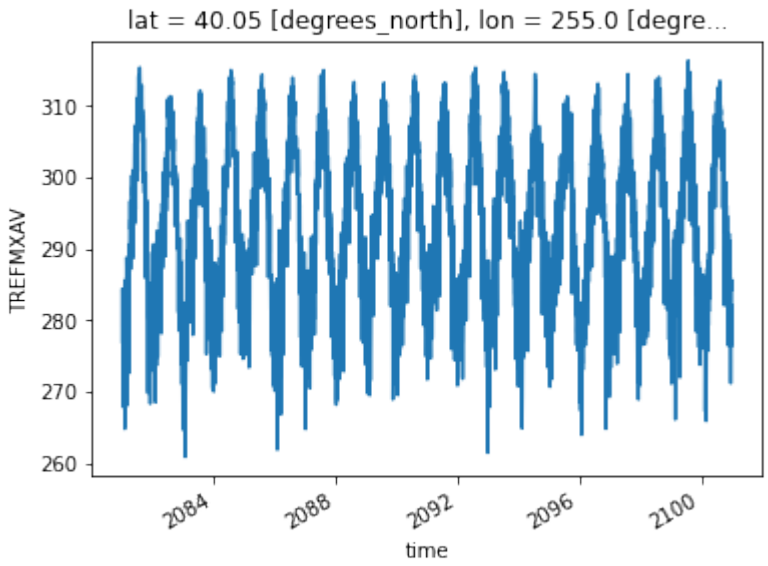
(continued from previous page)

```
1  279.335205  4.287326e-10  4.419851e-10
2  284.418121  1.492283e-15  7.843200e-14
3  283.729095  3.730888e-17  3.108414e-15
4  278.923859  5.766720e-11  9.267757e-11
```

data visualization

[4]:
```
ds["TREFMXAV"].plot()
```

[4]:
```
[<matplotlib.lines.Line2D at 0x2b747263b2b0>]
```



5.2 Step 2: automated machine learning

train a model (emulator)

[5]:
```
%%time
# assume that we want to split the data into training data and testing data
# let's use first 95% for training, and the remaining for testing
idx = df.shape[0]
train = df.iloc[:int(0.95*idx),:]
test = df.iloc[int(0.95*idx):,:]
(X_train, y_train) = (train[features], train[label].values)
(X_test, y_test) = (test[features], test[label].values)

# train the model
automl.fit(X_train=X_train, y_train=y_train,
           **automl_settings, verbose=-1)
print(automl.model.estimator)
```

```
LGBMRegressor(colsample_bytree=0.7463308378914483,
              learning_rate=0.1530612501227463, max_bin=1023,
              min_child_samples=2, n_estimators=60, num_leaves=49,
```

(continues on next page)

(continued from previous page)

```
              reg_alpha=0.0009765625, reg_lambda=0.012698515198279536,
              verbose=-1)
CPU times: user 3min 20s, sys: 2.53 s, total: 3min 22s
Wall time: 15.2 s
```

apply and test the machine learning model

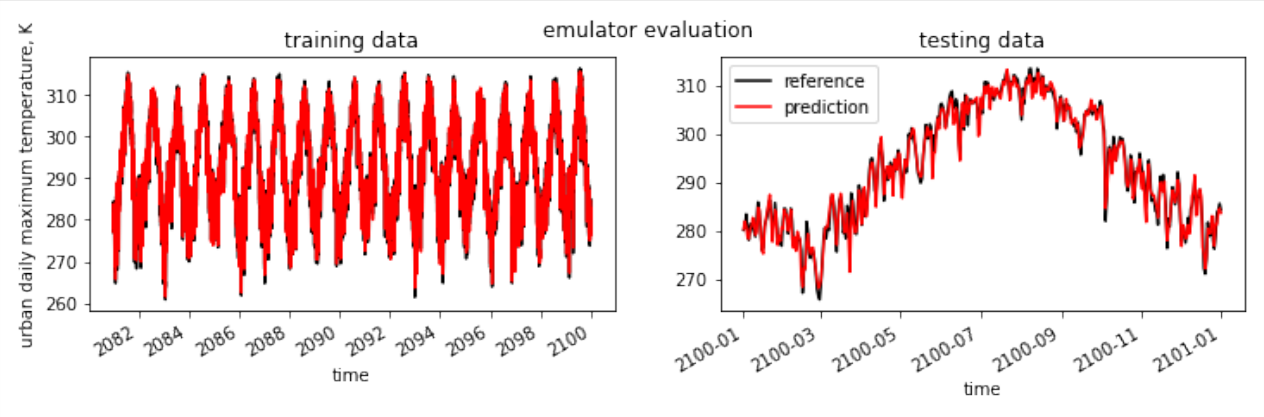
use `automl.predict(X)` to apply the model

[6]:
```
# training data
print("model performance using training data:")
y_pred = automl.predict(X_train)
print("root mean square error:",
      mean_squared_error(y_true=y_train, y_pred=y_pred, squared=False))
print("r2:", r2_score(y_true=y_train, y_pred=y_pred),"\n")
import pandas as pd
d_train = {"time":train["time"],"y_train":y_train.reshape(-1),"y_pred":y_pred.reshape(-
↪1)}
df_train = pd.DataFrame(d_train).set_index("time")

# testing data
print("model performance using testing data:")
y_pred = automl.predict(X_test)
print("root mean square error:",
      mean_squared_error(y_true=y_test, y_pred=y_pred, squared=False))
print("r2:", r2_score(y_true=y_test, y_pred=y_pred))
d_test = {"time":test["time"],"y_test":y_test.reshape(-1),"y_pred":y_pred.reshape(-1)}
df_test = pd.DataFrame(d_test).set_index("time")
```

```
model performance using training data:
root mean square error: 1.0953179201882188
r2: 0.9908044117105271

model performance using testing data:
root mean square error: 1.6714483065300212
r2: 0.9799441548983633
```

visualization

[7]:
```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(12,3))
fig.suptitle('emulator evaluation')
df_train["y_train"].plot(label="reference",c="k",ax=ax1)
df_train["y_pred"].plot(label="prediction",c="r",ax=ax1)
ax1.set_title("training data")
ax1.set_ylabel("urban daily maximum temperature, K")

df_test["y_test"].plot(label="reference",c="k",ax=ax2)
df_test["y_pred"].plot(label="prediction",c="r",ax=ax2)
ax2.set_title("testing data")
plt.legend()
plt.show()
```


emulator evaluation
training data
testing data
urban daily maximum temperature, K
310
300
290
280
270
260
2082
2084
2086
2088
2090
2092
2094
2096
2098
2100
time
reference
prediction
2100-01
2100-03
2100-05
2100-07
2100-09
2100-11
2101-01
time

CHAPTER

SIX

EXAMPLE FOR CMIP6

Here, CESM2 data serves as the training data, and the ML model trained on CESM2 data is applied to CMIP6 data.

Reference:
- GitHub: https://github.com/NCAR/cesm2-le-aws
- Data/Variables Information: - https://ncar.github.io/cesm2-le-aws/model_documentation.html#data-catalog
- https://registry.opendata.aws/cmip6/
- Reproduce CESM-LENS: https://github.com/NCAR/cesm2-le-aws/blob/main/notebooks/kay_et_al_lens2.ipynb
- Finding CMIP6 data using intake-esm and plotting time series for points by Zac Flamig

Step 0: load necessary packages and define parameters (no need to change)

[1]:
```
%%time
# Display output of plots directly in Notebook
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import json
import intake
from flaml import AutoML
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
import util
import math
import seaborn as sns

with open("./config_cesm2_cmip6.json",'r') as load_f:
#     param = json.loads(json.load(load_f))
    param = json.load(load_f)

    model = param["model"] # cesm2
    urban_type = param["urban_type"] # md
    city_loc = param["city_loc"] # {"lat": 40.0150, "lon": -105.2705}
    l_component = param["l_component"]
    a_component = param["a_component"]
    experiment = param["experiment"]
    frequency = param["frequency"]
```

(continues on next page)

(continued from previous page)

```
    cam_ls = param["cam_ls"]
    clm_ls = param["clm_ls"]
    forcing_variant = param["forcing_variant"]
    time = slice(param["time_start"],param["time_end"])
    member_id = param["member_id"]
    estimator_list = param["estimator_list"]
    time_budget = param["time_budget"]
    features = param["features"]
    label = param["label"]
    clm_var_mask = param["label"][0]
    CMIP6_url = param["CMIP6_url"]
    activity_id = param["activity_id"]
    experiment_id = param["experiment_id"]
    institution_id = param["institution_id"]
    table_id = param["table_id"]
```

```
CPU times: user 1.84 s, sys: 350 ms, total: 2.19 s
Wall time: 2.19 s
```

```
/glade/work/zhonghua/miniconda3/envs/aws_urban/lib/python3.8/site-packages/xgboost/
↪compat.py:31: FutureWarning: pandas.Int64Index is deprecated and will be removed from␣
↪pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

6.1 Step 1: load CESM2 data

[2]:
```
# get a dataset
ds = util.get_data(model, city_loc, experiment, frequency, member_id, time, cam_ls, clm_
↪ls,
                   forcing_variant=forcing_variant, urban_type=urban_type)
ds['time'] = ds.indexes['time'].to_datetimeindex()
```

```
--> The keys in the returned dictionary of datasets are constructed as follows:
        'component.experiment.frequency.forcing_variant'
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
different lat between CAM and CLM subgrid info, adjust subgrid info's lat
```

split into training and testing data

[3]:
```
mapping =  {
    "PRSN":"prsn",
    "PRECT":"pr",
    "PSL":"psl",
    "TREFHT":"tas",
    "TREFHTMN":"tasmin",
    "TREFHTMX":"tasmax"
}
```

(continues on next page)

(continued from previous page)

```
# create a dataframe
df = ds.to_dataframe().reset_index().dropna()
df["PRSN"] = (df["PRECSC"] + df["PRECSL"])*1000
df["PRECT"] = (df["PRECC"] + df["PRECL"])*1000

df_cesm = df.rename(columns=mapping)

# split the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(
    df_cesm[features], df_cesm[label], test_size=0.1, random_state=42)
display(X_train.head())
display(y_train.head())
```

```
                pr          prsn            psl         tas      tasmax  \
4253  5.120080e-05  1.210190e-18  102141.148438  297.131683  305.057770
712   8.689989e-06  4.476514e-06  102655.820312  273.535492  277.919128
4174  5.766846e-05  2.315565e-14  101171.445312  297.524048  305.175293
1811  8.906457e-07  7.931703e-07  101316.953125  271.924591  277.630005
5097  4.478946e-08  4.477884e-08  101735.164062  270.995270  275.632812

          tasmin
4253  291.910309
712   270.931824
4174  290.375336
1811  269.025482
5097  269.006317
```

```
        TREFMXAV
4253  305.300262
712   279.708527
4174  306.675201
1811  278.521729
5097  276.521973
```

6.2 Step 2: load CMIP6 data

[4]:
```
%%time
features = ['pr', 'prsn', 'psl','tas', 'tasmax', 'tasmin']

catalog = intake.open_esm_datastore('https://cmip6-pds.s3.amazonaws.com/pangeo-cmip6.json
↪')
catalog_subset = catalog.search(
    activity_id=activity_id,
    experiment_id=experiment_id,
    institution_id=institution_id,
    variable_id=features,
    table_id=table_id
)
datasets = catalog_subset.to_dataset_dict(zarr_kwargs={'consolidated': True, 'decode_
↪times': True})
datasets
```

```
--> The keys in the returned dictionary of datasets are constructed as follows:
        'activity_id.institution_id.source_id.experiment_id.table_id.grid_label'
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
CPU times: user 8.71 s, sys: 973 ms, total: 9.69 s
Wall time: 17 s
```

[4]:
```
{'ScenarioMIP.NOAA-GFDL.GFDL-ESM4.ssp370.day.gr1': <xarray.Dataset>
 Dimensions:    (bnds: 2, lat: 180, lon: 288, member_id: 1, time: 31390)
 Coordinates:
   * bnds       (bnds) float64 1.0 2.0
   * lat        (lat) float64 -89.5 -88.5 -87.5 -86.5 ... 86.5 87.5 88.5 89.5
     lat_bnds   (lat, bnds) float64 dask.array<chunksize=(180, 2), meta=np.ndarray>
   * lon        (lon) float64 0.625 1.875 3.125 4.375 ... 355.6 356.9 358.1 359.4
     lon_bnds   (lon, bnds) float64 dask.array<chunksize=(288, 2), meta=np.ndarray>
   * time       (time) object 2015-01-01 12:00:00 ... 2100-12-31 12:00:00
     time_bnds  (time, bnds) object dask.array<chunksize=(15695, 2), meta=np.ndarray>
   * member_id  (member_id) <U8 'r1i1p1f1'
     height     float64 2.0
 Data variables:
     pr         (member_id, time, lat, lon) float32 dask.array<chunksize=(1, 592, 180,␣
↪288), meta=np.ndarray>
     prsn       (member_id, time, lat, lon) float32 dask.array<chunksize=(1, 671, 180,␣
↪288), meta=np.ndarray>
     psl        (member_id, time, lat, lon) float32 dask.array<chunksize=(1, 452, 180,␣
↪288), meta=np.ndarray>
     tas        (member_id, time, lat, lon) float32 dask.array<chunksize=(1, 420, 180,␣
↪288), meta=np.ndarray>
     tasmax     (member_id, time, lat, lon) float32 dask.array<chunksize=(1, 420, 180,␣
↪288), meta=np.ndarray>
     tasmin     (member_id, time, lat, lon) float32 dask.array<chunksize=(1, 415, 180,␣
↪288), meta=np.ndarray>
 Attributes: (12/48)
     references:              see further_info_url attribute
     realm:                   atmos
     parent_time_units:       days since 1850-1-1
     sub_experiment:          none
     forcing_index:           1
     grid_label:              gr1
     ...                      ...
     parent_variant_label:    r1i1p1f1
     license:                 CMIP6 model data produced by NOAA-GFDL is licens...
     parent_mip_era:          CMIP6
     parent_activity_id:      CMIP
     title:                   NOAA GFDL GFDL-ESM4 model output prepared for CM...
     intake_esm_dataset_key:  ScenarioMIP.NOAA-GFDL.GFDL-ESM4.ssp370.day.gr1}
```

[5]:
```
%%time
# define the dataset name in the dictionary and the "member_id"
df_cmip = datasets['ScenarioMIP.NOAA-GFDL.GFDL-ESM4.ssp370.day.gr1']\
```

(continues on next page)

(continued from previous page)

```
          .sel(member_id = 'r1i1p1f1',
               time = slice(param["time_start"], param["time_end"]))\
          .sel(lat = param["city_loc"]["lat"],
               lon = util.lon_to_360(param["city_loc"]["lon"]),
               method="nearest")\
          [features].load()\
          .to_dataframe().reset_index()
df_cmip.head()
```

```
CPU times: user 20.3 s, sys: 11 s, total: 31.3 s
Wall time: 18.5 s
```

[5]:

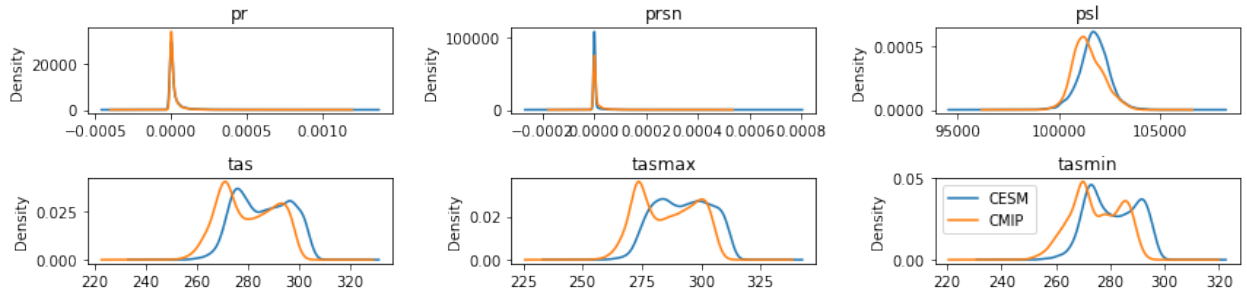
	time	pr	prsn	psl	tas	\
0	2081-01-02 12:00:00	4.284881e-11	1.210498e-13	101005.351562	267.040985	
1	2081-01-03 12:00:00	1.998000e-06	6.424573e-07	100812.898438	272.238739	
2	2081-01-04 12:00:00	1.482927e-05	8.858435e-06	100455.796875	273.498535	
3	2081-01-05 12:00:00	6.613790e-06	5.915619e-06	101060.375000	271.265442	
4	2081-01-06 12:00:00	9.097347e-06	6.341099e-06	100795.867188	272.010651	

	tasmax	tasmin	lat	lon	member_id	height
0	274.020386	264.221344	40.5	254.375	r1i1p1f1	2.0
1	275.491943	270.209229	40.5	254.375	r1i1p1f1	2.0
2	274.920624	271.907166	40.5	254.375	r1i1p1f1	2.0
3	273.210297	269.065277	40.5	254.375	r1i1p1f1	2.0
4	276.695343	269.123260	40.5	254.375	r1i1p1f1	2.0

6.3 Step 3: compare CESM2 training and CMIP6 data

[6]:

```
fig = plt.figure(figsize=(12,3))
idx = 1
for var in features:
    ax = fig.add_subplot(math.ceil(math.ceil(len(features)/3)), 3, idx)
    X_train[var].plot.kde(ax=ax, label="CESM")
    df_cmip[var].plot.kde(ax=ax, label="CMIP")
    idx+=1
    ax.set_title(var)
plt.legend()
plt.tight_layout()
plt.show()
```



6.4 Step 4: automated machine learning

train a model (emulator)

[7]:
```
%%time
# setup for automl
automl = AutoML()
automl_settings = {
    "time_budget": time_budget,  # in seconds
    "metric": 'r2',
    "task": 'regression',
    "estimator_list":estimator_list,
}

# train the model
automl.fit(X_train=X_train, y_train=y_train.values,
           **automl_settings, verbose=-1)
print(automl.model.estimator)
```

```
ExtraTreesRegressor(max_features=0.9591245274694429, max_leaf_nodes=426,
                    n_estimators=125, n_jobs=-1)
CPU times: user 2min 1s, sys: 2.9 s, total: 2min 4s
Wall time: 15.2 s
```

evaluate the model

[8]:
```
y_pred = automl.predict(X_test)
print("root mean square error:",
      round(mean_squared_error(y_true=y_test, y_pred=y_pred, squared=False),3))
print("r2:",
      round(r2_score(y_true=y_test, y_pred=y_pred),3))
```

```
root mean square error: 0.636
r2: 0.997
```

apply and test the machine learning model

use `automl.predict(X)` to apply the model

[9]:
```
df_cmip[label] = automl.predict(df_cmip[features]).reshape(df_cmip.shape[0],-1)
```

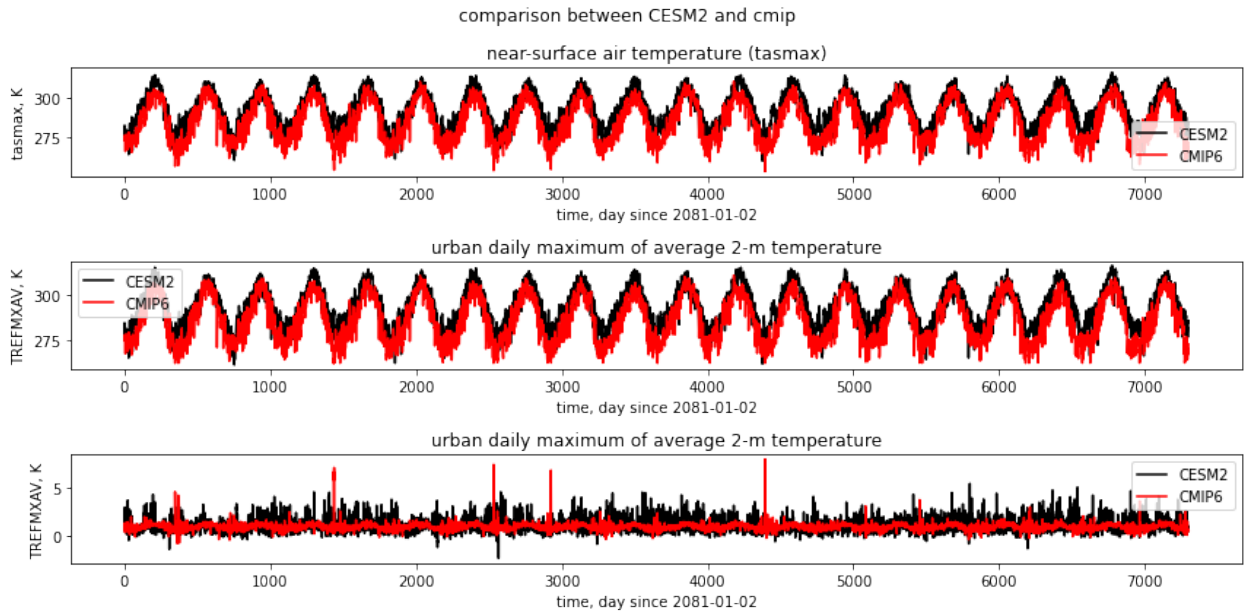
6.5 Step 5: visualization

[10]:
```
fig, (ax1,ax2,ax3) = plt.subplots(3,1,figsize=(12,6))
fig.suptitle('comparison between CESM2 and cmip')
df_cesm["tasmax"].plot(label="CESM2",c="k",ax=ax1)
df_cmip["tasmax"].plot(label="cmip",c="r",ax=ax1)
ax1.set_title("near-surface air temperature (tasmax)")
ax1.set_ylabel("tasmax, K")
ax1.set_xlabel("time, day since 2081-01-02")
```

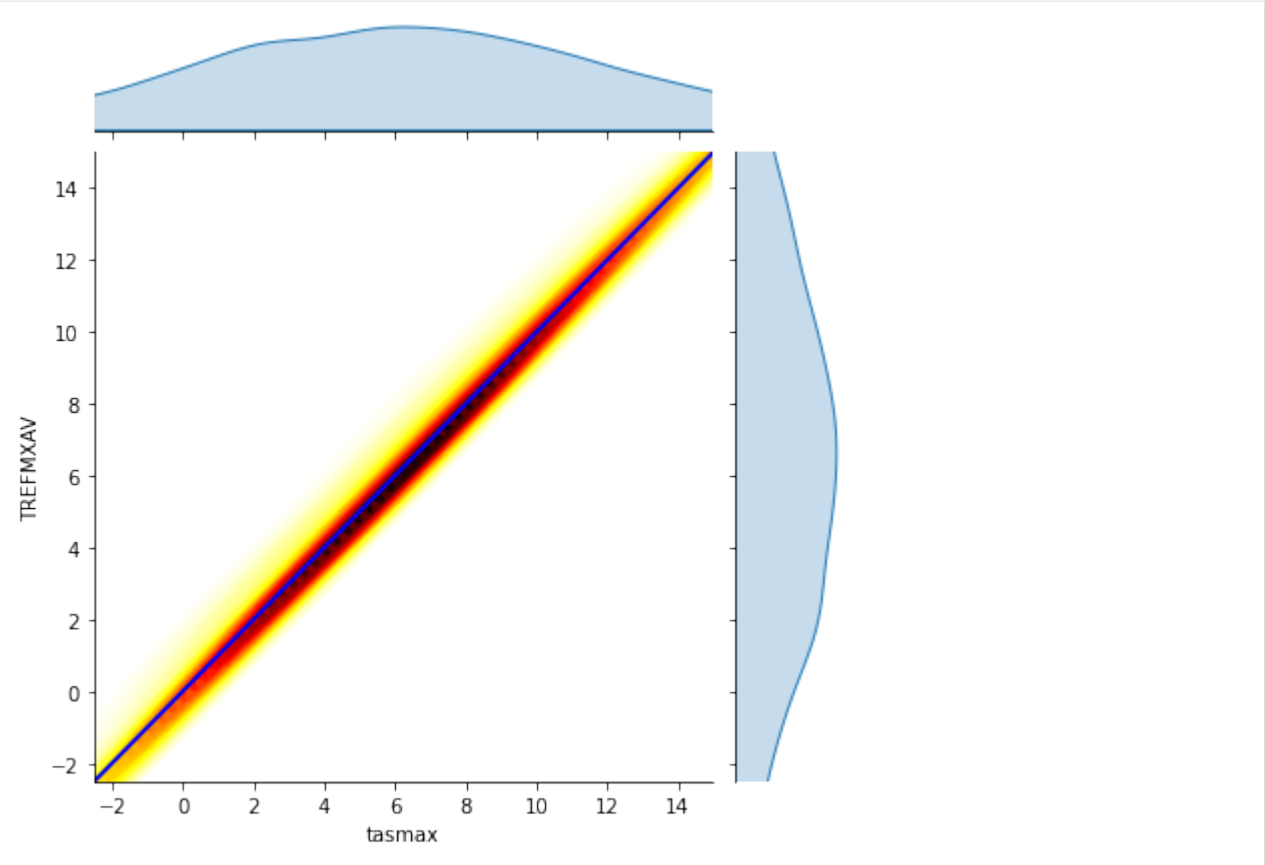
(continues on next page)

(continued from previous page)

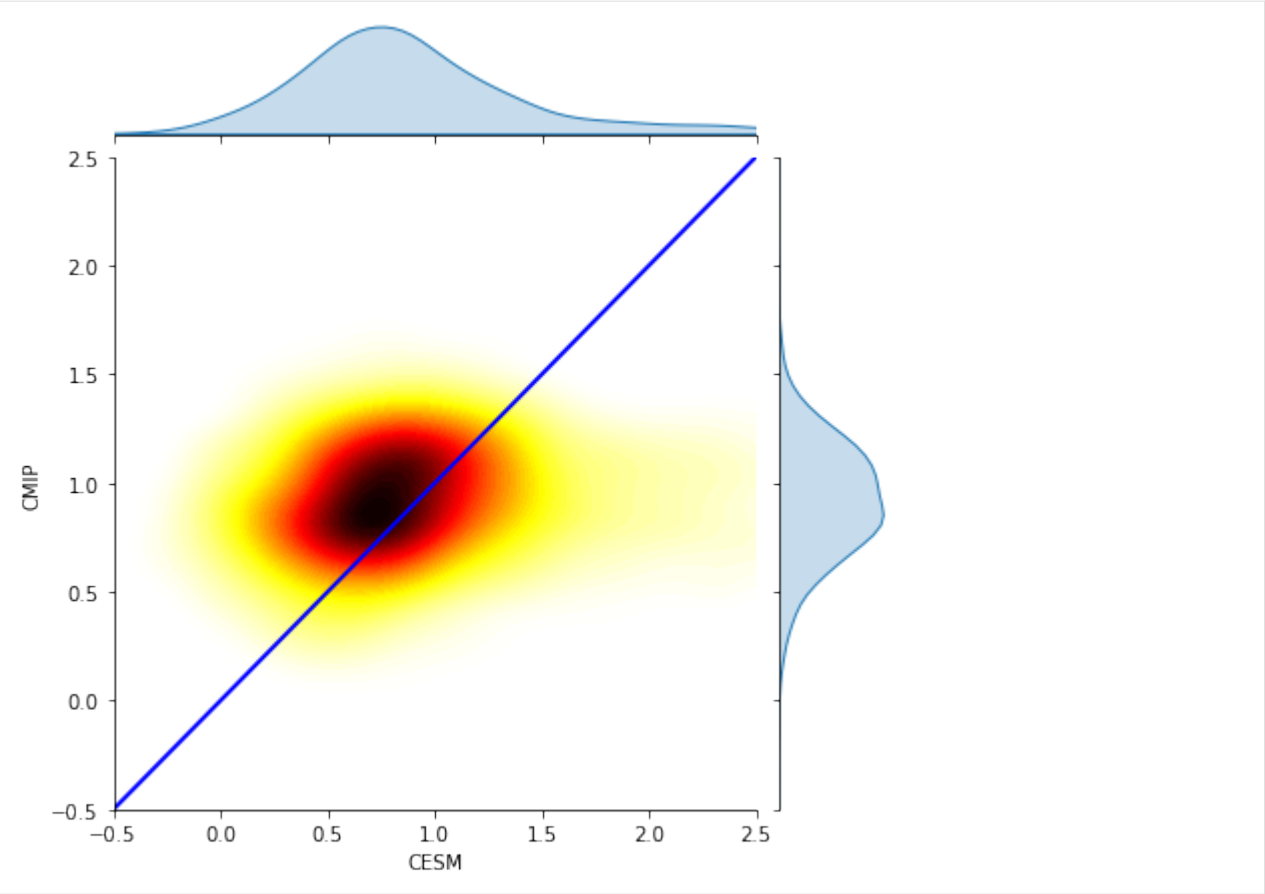
```
ax1.legend(["CESM2","CMIP6"])

df_cesm["TREFMXAV"].plot(label="CESM2",c="k",ax=ax2)
df_cmip["TREFMXAV"].plot(label="cmip",c="r",ax=ax2)
ax2.set_title("urban daily maximum of average 2-m temperature")
ax2.set_ylabel("TREFMXAV, K")
ax2.set_xlabel("time, day since 2081-01-02")
ax2.legend(["CESM2","CMIP6"])

(df_cesm["TREFMXAV"]-df_cesm["tasmax"]).plot(label="CESM2",c="k",ax=ax3)
(df_cmip["TREFMXAV"]-df_cmip["tasmax"]).plot(label="cmip",c="r",ax=ax3)
ax3.set_title("urban daily maximum of average 2-m temperature")
ax3.set_ylabel("TREFMXAV, K")
ax3.set_xlabel("time, day since 2081-01-02")
ax3.legend(["CESM2","CMIP6"])

plt.tight_layout()
plt.show()
```



[11]:
```
# reference: https://stackoverflow.com/questions/53964485/seaborn-jointplot-color-by-density
x = df_cesm["tasmax"]-df_cmip["tasmax"]
y = df_cesm["TREFMXAV"]-df_cmip["TREFMXAV"]

plot = sns.jointplot(x, y, kind="kde", cmap='hot_r', n_levels=60, fill=True)
plot.ax_joint.set_xlim(-2.5,15)
plot.ax_joint.set_ylim(-2.5,15)
plot.ax_joint.plot([-2.5,15], [-2.5,15], 'b-', linewidth = 2)
plt.show()
```



[12]:
```
# reference: https://stackoverflow.com/questions/53964485/seaborn-jointplot-color-by-
↪density
x = df_cesm["TREFMXAV"]-df_cesm["tasmax"]
y = df_cmip["TREFMXAV"]-df_cmip["tasmax"]

plot = sns.jointplot(x, y, kind="kde", cmap='hot_r', n_levels=60, fill=True)
plot.ax_joint.set_xlim(-0.5,2.5)
plot.ax_joint.set_ylim(-0.5,2.5)
plot.ax_joint.set_xlabel("CESM")
plot.ax_joint.set_ylabel("CMIP")
plot.ax_joint.plot([-0.5,2.5], [-0.5,2.5], 'b-', linewidth = 2)
plt.show()
```


2.5
2.0
1.5
1.0
0.5
0.0
−0.5
CMIP
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
CESM

CHAPTER

SEVEN

EXAMPLE FOR CESM2 CLIMATE CHANGE

Here, we load present (2016-01-02 to 2035-12-31) and future (2066-01-02 to 2085-12-31) data, and evaluate the applicability of a machine learning model trained on the present climate for predicting future urban climate.

NOTE: Compared to the CESM1 demo, here "Q" (QBOT), "U" (UBOT) and "V" (VBOT) are not included. When the bottom "lev" of "Q", "U", and "V" are merged, there is an issue.

Reference:
- GitHub: https://github.com/NCAR/cesm2-le-aws
- Data/Variables Information: https://ncar.github.io/cesm2-le-aws/model_documentation.html#data-catalog
- Reproduce CESM-LENS: https://github.com/NCAR/cesm2-le-aws/blob/main/notebooks/kay_et_al_lens2.ipynb

Step 0: load necessary packages and define parameters (no need to change)

[1]:
```
%%time
# Display output of plots directly in Notebook
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import json
from flaml import AutoML
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import math
import seaborn as sns
import util
import gc

import warnings
warnings.filterwarnings("ignore")
```

```
CPU times: user 1.76 s, sys: 338 ms, total: 2.1 s
Wall time: 2.1 s
```

```
/glade/work/zhonghua/miniconda3/envs/aws_urban/lib/python3.8/site-packages/xgboost/
↪compat.py:31: FutureWarning: pandas.Int64Index is deprecated and will be removed from␣
↪pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

7.1 Step 1: load future data (2066-01-02 to 2085-12-31)

[2]:
```
with open("./config_cesm2_climate_future.json",'r') as load_f:
#     param = json.loads(json.load(load_f))
    param = json.load(load_f)

    model = param["model"] # cesm2
    urban_type = param["urban_type"] # md
    city_loc = param["city_loc"] # {"lat": 40.0150, "lon": -105.2705}
    l_component = param["l_component"]
    a_component = param["a_component"]
    experiment = param["experiment"]
    frequency = param["frequency"]
    cam_ls = param["cam_ls"]
    clm_ls = param["clm_ls"]
    forcing_variant = param["forcing_variant"]
    time = slice(param["time_start"],param["time_end"])
    member_id = param["member_id"]
#     estimator_list = param["estimator_list"]
#     time_budget = param["time_budget"]
    features = param["features"]
    label = param["label"]
    clm_var_mask = param["label"][0]

# get a dataset
ds = util.get_data(model, city_loc, experiment, frequency, member_id, time, cam_ls, clm_
↪ls,
                   forcing_variant=forcing_variant, urban_type=urban_type)

# create a dataframe
ds['time'] = ds.indexes['time'].to_datetimeindex()
df_future = ds.to_dataframe().reset_index().dropna()

if "PRSN" in features:
    df_future["PRSN"] = df_future["PRECSC"] + df_future["PRECSL"]
if "PRECT" in features:
    df_future["PRECT"] = df_future["PRECC"] + df_future["PRECL"]

X_future_train, X_future_test, y_future_train, y_future_test = train_test_split(
    df_future[features], df_future[label], test_size=0.05, random_state=66)
display(X_future_train.head())
display(y_future_train.head())

del ds
gc.collect()
```

```
--> The keys in the returned dictionary of datasets are constructed as follows:
        'component.experiment.frequency.forcing_variant'
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
different lat between CAM and CLM subgrid info, adjust subgrid info's lat
```

```
            FLNS        FSNS         PRECT          PRSN      TREFHT  \
3141  132.101913  282.795959  1.133692e-18  1.683445e-22  301.627167
3374  102.263107  232.231064  1.288310e-09  3.225936e-17  282.054474
4966   84.371529  157.059128  4.308776e-09  5.351772e-24  301.114380
4898  112.747520  312.061737  3.106702e-12  1.646209e-16  293.713379
257   103.962410  219.334702  1.838627e-12  2.581992e-19  293.866547

        TREFHTMX    TREFHTMN
3141  308.214355  294.433472
3374  290.421173  275.131805
4966  308.091553  295.771759
4898  300.738068  286.623810
257   303.037964  285.171997
```

```
        TREFMXAV
3141  308.828857
3374  291.491394
4966  308.434662
4898  302.386292
257   303.614197
```

[2]:
```
1157
```

7.2 Step 2: load present data (2016-01-02 to 2035-12-31)

[3]:
```
with open("./config_cesm2_climate_present.json",'r') as load_f:
#     param = json.loads(json.load(load_f))
    param = json.load(load_f)

    model = param["model"] # cesm2
    urban_type = param["urban_type"] # md
    city_loc = param["city_loc"] # {"lat": 40.0150, "lon": -105.2705}
    l_component = param["l_component"]
    a_component = param["a_component"]
    experiment = param["experiment"]
    frequency = param["frequency"]
    cam_ls = param["cam_ls"]
    clm_ls = param["clm_ls"]
    forcing_variant = param["forcing_variant"]
    time = slice(param["time_start"],param["time_end"])
    member_id = param["member_id"]
    estimator_list = param["estimator_list"]
    time_budget = param["time_budget"]
    features = param["features"]
    label = param["label"]
    clm_var_mask = param["label"][0]

# get a dataset
ds = util.get_data(model, city_loc, experiment, frequency, member_id, time, cam_ls, clm_
↪ls,
```

(continues on next page)

(continued from previous page)

```
                    forcing_variant=forcing_variant, urban_type=urban_type)

# create a dataframe
ds['time'] = ds.indexes['time'].to_datetimeindex()
df_present = ds.to_dataframe().reset_index().dropna()

if "PRSN" in features:
    df_present["PRSN"] = df_present["PRECSC"] + df_present["PRECSL"]
if "PRECT" in features:
    df_present["PRECT"] = df_present["PRECC"] + df_present["PRECL"]

X_present_train, X_present_test, y_present_train, y_present_test = train_test_split(
    df_present[features], df_present[label], test_size=0.05, random_state=66)
display(X_present_train.head())
display(y_present_train.head())

del ds
gc.collect()
```

```
--> The keys in the returned dictionary of datasets are constructed as follows:
        'component.experiment.frequency.forcing_variant'
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
different lat between CAM and CLM subgrid info, adjust subgrid info's lat
```

	FLNS	FSNS	PRECT	PRSN	TREFHT	\
3141	68.676613	199.265472	1.852086e-08	2.289983e-20	291.946472	
3374	94.056053	220.666367	6.313760e-09	6.944248e-15	277.478851	
4966	114.924210	275.390778	8.211864e-11	1.427359e-19	298.891174	
4898	121.778847	326.139313	1.517838e-10	8.360884e-20	288.308014	
257	89.618881	217.455154	1.282198e-11	1.113258e-15	283.563782	

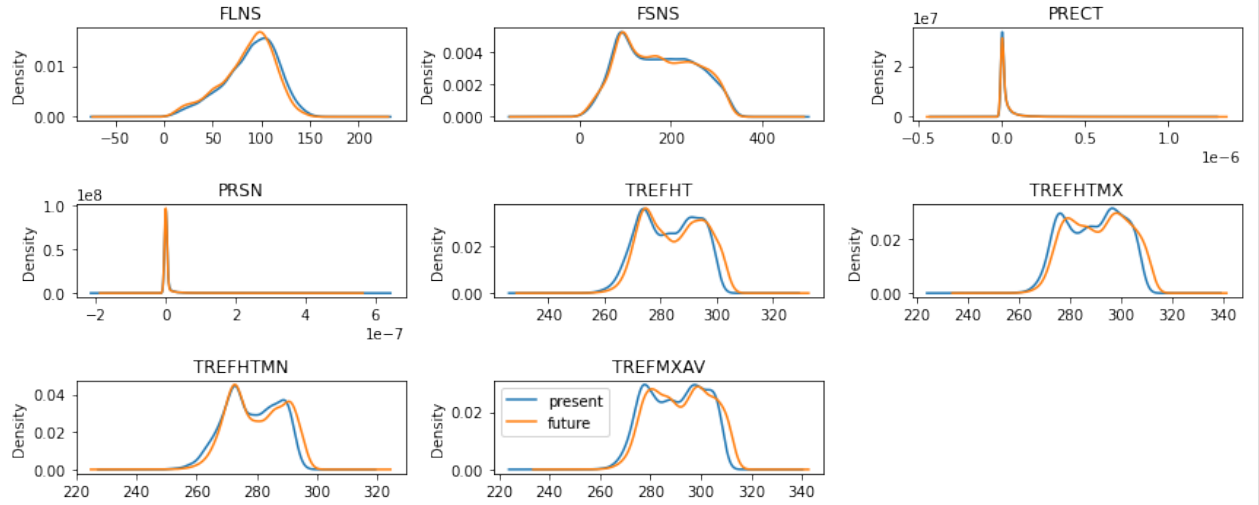
	TREFHTMX	TREFHTMN
3141	300.030487	287.180969
3374	284.147064	274.174713
4966	307.201935	290.399048
4898	294.743103	280.171753
257	295.233582	277.655365

	TREFMXAV
3141	300.976105
3374	285.210724
4966	307.383545
4898	296.533691
257	296.322083

[3]: 1503

7.3 Step 3: compare future and present training data

[4]:
```
fig = plt.figure(figsize=(12,5))
idx = 1
for var in features:
    ax = fig.add_subplot(math.ceil(math.ceil(len(features)/3)), 3, idx)
    X_present_train[var].plot.kde(ax=ax)
    X_future_train[var].plot.kde(ax=ax)
    idx+=1
    ax.set_title(var)

var = "TREFMXAV"
ax = fig.add_subplot(math.ceil(math.ceil(len(features)/3)), 3, idx)
y_present_train[var].plot.kde(ax=ax, label="present")
y_future_train[var].plot.kde(ax=ax, label="future")
ax.set_title(var)
plt.legend()

plt.tight_layout()
plt.show()
```



7.4 Step 4: automated machine learning

train a model (emulator)

[5]:
```
%%time
# setup for automl
automl = AutoML()
automl_settings = {
    "time_budget": time_budget,  # in seconds
    "metric": 'r2',
    "task": 'regression',
    "estimator_list":estimator_list,
}
```

(continues on next page)

(continued from previous page)

```
# train the model
automl.fit(X_train=X_present_train, y_train=y_present_train.values,
           **automl_settings, verbose=-1)
print(automl.model.estimator)

# evaluate the model
y_present_pred = automl.predict(X_present_test)
print("root mean square error:",
      round(mean_squared_error(y_true=y_present_test, y_pred=y_present_pred,␣
↪squared=False),3))
print("r2:",
      round(r2_score(y_true=y_present_test, y_pred=y_present_pred),3))
```

```
LGBMRegressor(learning_rate=0.07667973275997925, max_bin=1023,
              min_child_samples=2, n_estimators=141, num_leaves=11,
              reg_alpha=0.006311706639004055, reg_lambda=0.048417038177217056,
              verbose=-1)
root mean square error: 0.53
r2: 0.998
CPU times: user 7min 14s, sys: 7.27 s, total: 7min 21s
Wall time: 30.1 s
```

apply the model to future climate and evaluate

[6]:
```
y_future_pred = automl.predict(X_future_test)
print("root mean square error:",
      round(mean_squared_error(y_true=y_future_test, y_pred=y_future_pred,␣
↪squared=False),3))
print("r2:",
      round(r2_score(y_true=y_future_test, y_pred=y_future_pred),3))
```

```
root mean square error: 0.741
r2: 0.995
```

use the future climate data to train and evaluate

[7]:
```
%%time
# setup for automl
automl = AutoML()
automl_settings = {
    "time_budget": time_budget,  # in seconds
    "metric": 'r2',
    "task": 'regression',
    "estimator_list":estimator_list,
}

# train the model
automl.fit(X_train=X_future_train, y_train=y_future_train.values,
           **automl_settings, verbose=-1)
print(automl.model.estimator)

# evaluate the model
```

(continues on next page)

(continued from previous page)

```
y_future_pred = automl.predict(X_future_test)
print("root mean square error:",
      round(mean_squared_error(y_true=y_future_test, y_pred=y_future_pred,␣
↪squared=False),3))
print("r2:",
      round(r2_score(y_true=y_future_test, y_pred=y_future_pred),3))
```

```
ExtraTreesRegressor(max_features=0.926426234471867, max_leaf_nodes=501,
                    n_estimators=119, n_jobs=-1)
root mean square error: 0.665
r2: 0.996
CPU times: user 4min 50s, sys: 7 s, total: 4min 57s
Wall time: 30.2 s
```

CHAPTER

EIGHT

HOW TO CREATE A JSON FILE?

JSON is an open standard file format an data interchange format that uses human-readable text to store and transmit data object.

- You can click here to find more information of JSON.
- Please check here for CESM1 variables and here for CESM2 variables

```
{
    "model": "cesm1",
    "urban_type": "md",
    "city_loc": {"lat": 40.1164, "lon": -88.2434},
    "l_component": "lnd",
    "a_component": "atm",
    "experiment": "RCP85",
    "frequency": "daily",
    "cam_ls": ["TREFHT", "TREFHTMX", "FLNS", "FSNS", "PRECSC", "PRECSL", "PRECT", "QBOT",
↪ "UBOT", "VBOT"],
    "clm_ls": ["TREFMXAV_U"],
    "forcing_variant": "cmip6",
    "time_start": "2081-01-02",
    "time_end": "2100-12-31",
    "member_id": [2],
    "estimator_list": ["lgbm", "xgboost", "rf", "extra_tree"],
    "time_budget": 15,
    "features": ["FLNS", "FSNS", "PRECT", "PRSN", "QBOT", "TREFHT", "UBOT", "VBOT"],
    "label": ["TREFMXAV_U"]
}
```

where

```
model : string
    "cesm1" or "cesm2"
urban_type : string
    e.g., "tbd" (Tall Building District), "hd" (High Density), and "md" (Medium Density)
city_loc : dict
    a dict of a city's lat and lon that we are interested in , e.g., {'lat': 40.1164,
↪'lon': -88.2434}
l_component : str, optional
    component name of CLM, by default "lnd"
a_component : str, optional
    component name of CAM, by default "atm"
experiment : string
```

(continues on next page)

(continued from previous page)

```
    e.g., "RCP85" (RCP 8.5 runs)
frequency : string
    e.g., "daily" or "monthly"
cam_ls : a list of string
    CAM (atmospheric forcing) variables, e.g., ["TREFHT", "TREFHTMX", "FLNS", "FSNS"]
clm_ls : a list of string
    CLM (land) variables, e.g., ["TREFMXAV_U"] (Urban daily maximum of average 2-m␣
↪temperature)
forcing_variant : string
    the biomass forcing variant, e.g.,
    "cmip6" (the default in the cmip6 runs),
    "smbb" (smoothed biomass burning)
time_start: string
    start date, e.g., "2081-01-02"
time_end: string
    end date, e.g., "2100-12-31"
member_id : a list of int
    CESM1 large ensemble member ID, e.g., [2,3]
estimator_list : a list of string
    a list of strings for estimator names, e.g., ["lgbm", "xgboost", "rf", "extra_tree"],
↪ or 'auto'
time_budget : int
    total running time in seconds, e.g., 15
features : a list of string
    features (predictors) for machine learning, it should be a subset of "cam_ls"
label: a list of string (currently we only support a single element within the list)
    label for machine learning, "label" means something we want to predict, e.g., [
↪"TREFMXAV_U"]
```

How to save a JSON file using Python

Note: please pay attention to the difference between CESM1 and CESM2

[1]:
```
import json

# CESM1
cesm1 = {
    "model": "cesm1",
    "urban_type": "md",
    "city_loc": {"lat": 40.1164, "lon": -88.2434},
    "l_component": "lnd",
    "a_component": "atm",
    "experiment": "RCP85",
    "frequency": "daily",
    "cam_ls": ["TREFHT", "TREFHTMX", "FLNS", "FSNS", "PRECSC", "PRECSL", "PRECT", "QBOT",
↪ "UBOT", "VBOT"],
    "clm_ls": ["TREFMXAV_U"],
    "forcing_variant": "cmip6",
    "time_start": "2081-01-02",
    "time_end": "2100-12-31",
    "member_id": [2],
    "estimator_list": ["lgbm", "xgboost", "rf", "extra_tree"],
    "time_budget": 15,
```

(continues on next page)

(continued from previous page)

```
    "features": ["FLNS", "FSNS", "PRECT", "PRSN", "QBOT", "TREFHT", "UBOT", "VBOT"],
    "label": ["TREFMXAV_U"]
}

# CESM2
cesm2 = {
    "model": "cesm2",
    "urban_type": "md",
    "city_loc": {"lat": 40.1164, "lon": -88.2434},
    "l_component": "lnd",
    "a_component": "atm",
    "experiment": "ssp370",
    "frequency": "daily",
    "cam_ls": ["TREFHT", "TREFHTMX", "FLNS", "FSNS", "PRECSC", "PRECSL", "PRECC", "PRECL
↪"],
    "clm_ls": ["TREFMXAV"],
    "forcing_variant": "cmip6",
    "time_start": "2081-01-02",
    "time_end": "2100-12-31",
    "member_id": ["r1i1231p1f1"],
    "estimator_list": ["lgbm", "xgboost", "rf", "extra_tree"],
    "time_budget": 15,
    "features": ["FLNS", "FSNS", "PRECT", "PRSN", "TREFHT"],
    "label": ["TREFMXAV"]
}

with open("./config_cesm1.json", "w") as outfile:
    json.dump(cesm1, outfile, indent=4)

with open("./config_cesm2.json", "w") as outfile:
    json.dump(cesm2, outfile, indent=4)
```

How to load a JSON file using Python

[2]:
```
# CESM 1
with open("./config_cesm1.json",'r') as load_f:
    param = json.load(load_f)
param
```

[2]:
```
{'model': 'cesm1',
 'urban_type': 'md',
 'city_loc': {'lat': 40.1164, 'lon': -88.2434},
 'l_component': 'lnd',
 'a_component': 'atm',
 'experiment': 'RCP85',
 'frequency': 'daily',
 'cam_ls': ['TREFHT',
  'TREFHTMX',
  'FLNS',
  'FSNS',
  'PRECSC',
  'PRECSL',
  'PRECT',
```

(continues on next page)

(continued from previous page)

```
 'QBOT',
 'UBOT',
 'VBOT'],
'clm_ls': ['TREFMXAV_U'],
'forcing_variant': 'cmip6',
'time_start': '2081-01-02',
'time_end': '2100-12-31',
'member_id': [2],
'estimator_list': ['lgbm', 'xgboost', 'rf', 'extra_tree'],
'time_budget': 15,
'features': ['FLNS',
 'FSNS',
 'PRECT',
 'PRSN',
 'QBOT',
 'TREFHT',
 'UBOT',
 'VBOT'],
'label': ['TREFMXAV_U']}
```

CHAPTER

NINE

HOW TO CREATE A MASK FOR CESM1'S "URBAN AREAS"?

This script is used for creating a urban mask at the global scale for CESM1 data.

Reference:
- GitHub: https://github.com/ncar/cesm-lens-aws/
- (outdated) Reproduce CESM-LENS:
http://gallery.pangeo.io/repos/NCAR/cesm-lens-aws/notebooks/kay-et-al-2015.v3.html

Step 0: load necessary packages and define parameters

[1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import intake
import numpy as np
import pandas as pd
import xarray as xr
import matplotlib.pyplot as plt

# define parameters for data retrieval
catalog_url = 'https://raw.githubusercontent.com/NCAR/cesm-lens-aws/main/intake-catalogs/
↪aws-cesm1-le.json'
experiment = "RCP85"
frequency = "daily"
urban_variable = "TREFMXAV_U"
cam_variable = "TREFHT"
```

Step 1: load datasets

[2]:
```
col = intake.open_esm_datastore(catalog_url)
col_subset = col.search(experiment=experiment, frequency=frequency, variable=urban_
↪variable)
dsets = col_subset.to_dataset_dict(zarr_kwargs={"consolidated": True},
                                   storage_options={"anon": True})["lnd.RCP85.daily"]
```

```
--> The keys in the returned dictionary of datasets are constructed as follows:
        'component.experiment.frequency'
```

```
<IPython.core.display.HTML object>
```

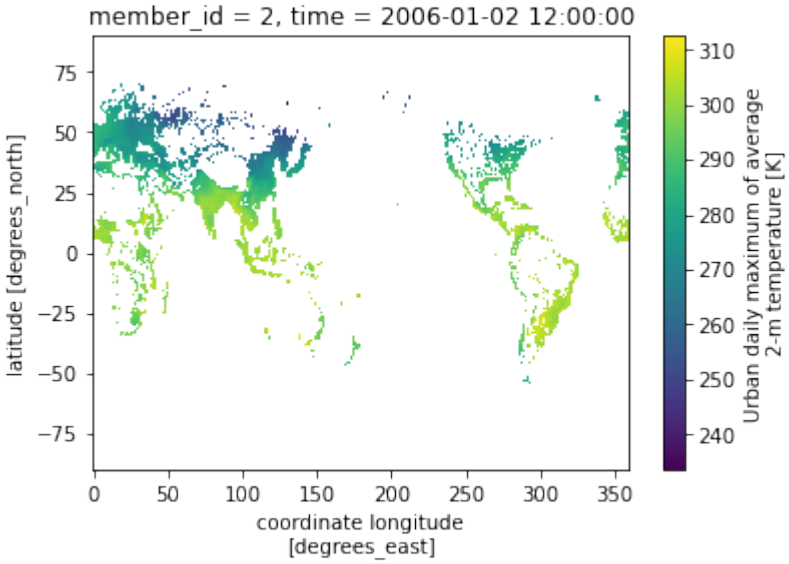
```
<IPython.core.display.HTML object>
```

Step 2: find the urban gridcell

Given that urban gridcell is **time-invariant**, let's use `member_id = 2` and `time="2006-01-02"`

[3]:
```
da = dsets.sel(member_id=2, time="2006-01-02")[urban_variable].load()
da.plot()
```

[3]:
```
<matplotlib.collections.QuadMesh at 0x2abf60495790>
```



Step 3: save the urban mask

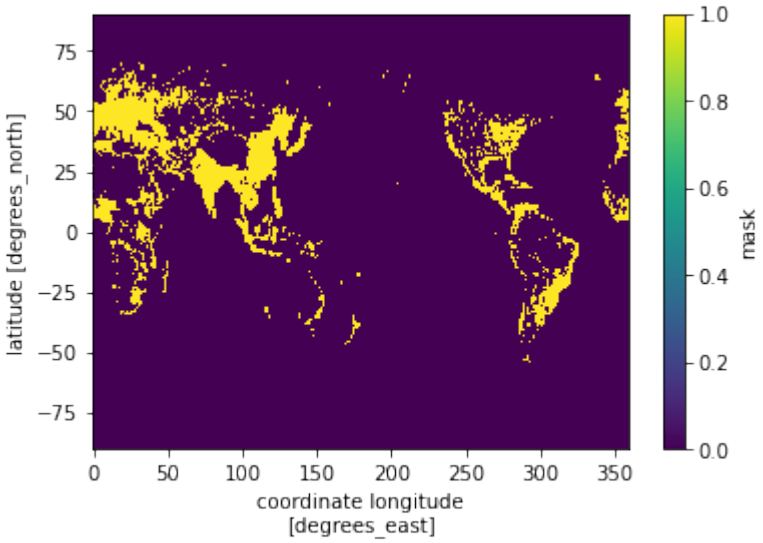
The file is save at current working directory, with a file name "urban_mask.nc"

[4]:
```
da.notnull().squeeze().drop(["time","member_id"]).rename("mask").to_netcdf("./CESM1_
↪urban_mask.nc")
```

Step 4: load the urban mask

[5]:
```
mask = xr.open_dataset("./CESM1_urban_mask.nc")["mask"]
mask.plot()
```

[5]:
```
<matplotlib.collections.QuadMesh at 0x2abf60a2df10>
```



Step 5: apply the urban mask to CAM

load CAM data

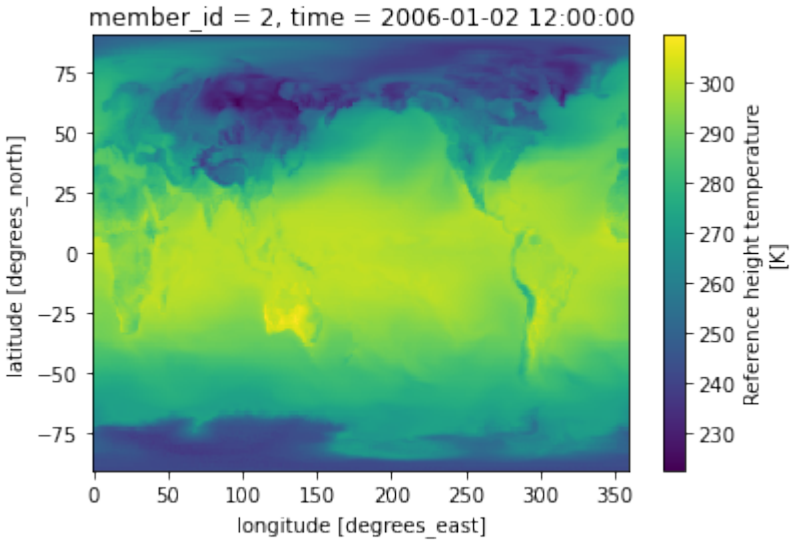
[6]:
```
col_subset = col.search(experiment=experiment, frequency=frequency, variable=cam_
↪variable)
dsets = col_subset.to_dataset_dict(zarr_kwargs={"consolidated": True},
                                   storage_options={"anon": True})['atm.RCP85.daily']
da_cam = dsets.sel(member_id=2, time="2006-01-02")[cam_variable].load()
da_cam.plot()
```

```
--> The keys in the returned dictionary of datasets are constructed as follows:
        'component.experiment.frequency'
```

```
<IPython.core.display.HTML object>
```

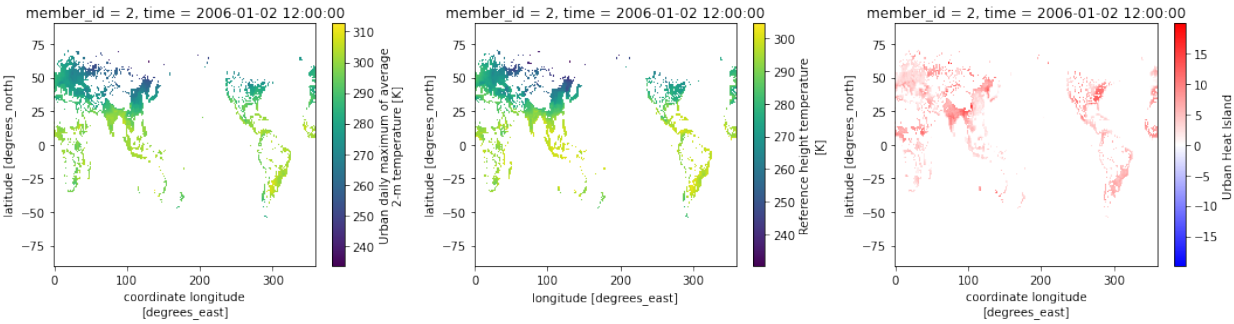
```
<IPython.core.display.HTML object>
```

[6]:
```
<matplotlib.collections.QuadMesh at 0x2abf6164fd60>
```



apply the mask to CAM data and calculate the difference

[7]:
```
da_cam_urban = da_cam.where(mask)

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,4))
da.plot(ax=ax1)
da_cam_urban.plot(ax=ax2)
(da-da_cam_urban).rename("Urban Heat Island").plot(ax=ax3, cmap="bwr")
plt.tight_layout()
```



check the dimension

[8]:
```
print("city number:", da.to_dataframe().dropna().shape[0])
assert (da-da_cam_urban).rename("Urban Heat Island").to_dataframe().dropna().shape[0] ==␣
↪4439
```

```
city number: 4439
```

CHAPTER

TEN

HOW TO CREATE A SUBGRID INFO FILE FOR CESM2'S CLM PROCESSING?

Why we need this script to save the a "`CESM2_subgrid_info.nc`" for CESM CLM processing?
Because the CLM files uploaded to AWS only contain:
- dimension: `(member_id, time, landunit)`
- coordinates: `member_id`, and `time`

In general, CESM's CLM varibales are saved as `(time, landunit)`.
But usually we want to analyze the datasets with the format `(time, lat, lon)`.
So the workflow for CESM's CLM variables would be - `(:, landunit)` [1D] -> `(:, landtype, lat, lon)` [3D] -> `(:, lat, lon)` [2D]

As a result, we at least need to know:
- `land1d_ixy` (landunit): for mapping from 1D to 3D
- `land1d_jxy` (landunit): for mapping from 1D to 3D
- `land1d_ityplunit` (landunit): for mapping from 1D to 3D
- `lat`: for setting up the list of lat
- `lon`: for setting up the list of lon

reference: https://github.com/zzheng93/CLM-1D-to-2D

[1]:
```
#https://github.com/NCAR/ctsm_python_gallery/blob/master/notebooks/PFT-Gridding.ipynb
import numpy as np
import xarray as xr

class load_clm:
    def __init__(self, args):
        self.ds = xr.open_dataset(args)
        self.lat = self.ds.lat
        self.lon = self.ds.lon
        self.time = self.ds.time
        self.ixy = self.ds.land1d_ixy
        self.jxy = self.ds.land1d_jxy
        self.ltype = self.ds.land1d_ityplunit
        self.ltype_dict = {value:key for key, value in self.ds.attrs.items() if 'ltype_'␣
↪in key.lower()}
```

(continues on next page)

(continued from previous page)

```
    def get2D(self, var_str):
        var = self.ds[var_str]
        nlat = len(self.lat.values)
        nlon = len(self.lon.values)
        ntim = len(self.time.values)
        nltype = len(self.ltype_dict)
        # create an empty array
        gridded = np.full([ntim,nltype,nlat,nlon],np.nan)
        # assign the values
        gridded[:,
                self.ltype.values.astype(int) - 1, # Fortran arrays start at 1
                self.jxy.values.astype(int) - 1,
                self.ixy.values.astype(int) - 1] = var.values
        grid_dims = xr.DataArray(gridded, dims=("time","ltype","lat","lon"))
        grid_dims = grid_dims.assign_coords(time=self.time,
                                            ltype=[i for i in range(self.ltype.values.
↪min(),
                                                                    self.ltype.values.
↪max()+1)],
                                            lat=self.lat.values,
                                            lon=self.lon.values)
        grid_dims.name = var_str
        return grid_dims
```

[2]:
```
fp = "/glade/campaign/cgd/cesm/CESM2-LE/timeseries/lnd/proc/tseries/day_1/TREFMXAV/"
fn = "b.e21.BSSP370smbb.f09_g17.LE2-1281.019.clm2.h6.TREFMXAV.20950101-21001231.nc"
clm = load_clm(fp+fn)
clm.ltype_dict
```

[2]:
```
{1: 'ltype_vegetated_or_bare_soil',
 2: 'ltype_crop',
 3: 'ltype_UNUSED',
 4: 'ltype_landice_multiple_elevation_classes',
 5: 'ltype_deep_lake',
 6: 'ltype_wetland',
 7: 'ltype_urban_tbd',
 8: 'ltype_urban_hd',
 9: 'ltype_urban_md'}
```

[3]:
```
clm.ds[["lat","lon",
        "land1d_ixy","land1d_jxy","land1d_ityplunit",
        "land1d_lon","land1d_lat",
        "landfrac","landmask","land1d_wtgcell","land1d_active"]].to_netcdf("./CESM2_
↪subgrid_info.nc")
clm.ds
```

[3]:
```
<xarray.Dataset>
Dimensions:             (levgrnd: 25, levlak: 10, levdcmp: 25, lon: 288,
                         lat: 192, gridcell: 21013, landunit: 62125,
                         column: 554298, pft: 848480, time: 2191,
                         hist_interval: 2)
Coordinates:
```

(continues on next page)

(continued from previous page)

```
  * levgrnd             (levgrnd) float32 0.01 0.04 0.09 ... 19.48 28.87 42.0
  * levlak              (levlak) float32 0.05 0.6 2.1 4.6 ... 25.6 34.33 44.78
  * levdcmp             (levdcmp) float32 0.01 0.04 0.09 ... 19.48 28.87 42.0
  * lon                 (lon) float32 0.0 1.25 2.5 3.75 ... 356.2 357.5 358.8
  * lat                 (lat) float32 -90.0 -89.06 -88.12 ... 88.12 89.06 90.0
  * time                (time) object 2095-01-01 00:00:00 ... 2101-01-01 00:0...
Dimensions without coordinates: gridcell, landunit, column, pft, hist_interval
Data variables: (12/45)
    area                (lat, lon) float32 ...
    landfrac            (lat, lon) float32 ...
    landmask            (lat, lon) float64 ...
    pftmask             (lat, lon) float64 ...
    nbedrock            (lat, lon) float64 ...
    grid1d_lon          (gridcell) float64 ...
    ...                  ...
    mscur               (time) int32 ...
    nstep               (time) int32 ...
    time_bounds         (time, hist_interval) object ...
    date_written        (time) |S16 ...
    time_written        (time) |S16 ...
    TREFMXAV            (time, landunit) float32 ...
Attributes: (12/102)
    title:                                  CLM History file information
    comment:                                NOTE: None of the variables ar...
    Conventions:                            CF-1.0
    history:                                created on 02/01/21 23:11:01
    source:                                 Community Land Model CLM4.0
    hostname:                               aleph
    ...                                     ...
    cft_irrigated_tropical_corn:            62
    cft_tropical_soybean:                   63
    cft_irrigated_tropical_soybean:         64
    time_period_freq:                       day_1
    Time_constant_3Dvars_filename:          ./b.e21.BSSP370smbb.f09_g17.LE...
    Time_constant_3Dvars:                   ZSOI:DZSOI:WATSAT:SUCSAT:BSW:H...
```

[4]:
```
clm.ds.landunit
```

[4]:
```
<xarray.DataArray 'landunit' (landunit: 62125)>
array([    0,     1,     2, ..., 62122, 62123, 62124])
Dimensions without coordinates: landunit
```

[5]:
```
clm.ds.land1d_ixy
```

[5]:
```
<xarray.DataArray 'land1d_ixy' (landunit: 62125)>
array([  1,   1,   1, ..., 265, 265, 265], dtype=int32)
Dimensions without coordinates: landunit
Attributes:
    long_name:  2d longitude index of corresponding landunit
```

[6]:
```
clm.ds.land1d_jxy
```

[6]:
```
<xarray.DataArray 'land1d_jxy' (landunit: 62125)>
array([  1,   1,   1, ..., 186, 186, 186], dtype=int32)
Dimensions without coordinates: landunit
Attributes:
    long_name:  2d latitude index of corresponding landunit
```

CHAPTER

ELEVEN

HOW TO ASK FOR HELP?

The GitHub issue tracker is the primary place for bug reports.

CHAPTER

TWELVE

ACKNOWLEDGMENTS

We thank AWS for providing AWS Cloud Credits for Research.

We would like to acknowledge high-performance computing support from Cheyenne (doi:10.5065/D6RX99HX) provided by NCAR's Computational and Information Systems Laboratory, sponsored by the National Science Foundation.