# Urban Climate Explorer

***Release 0.0.0***

**Zhonghua Zheng**

**Apr 20, 2022**

# OVERVIEW

Explore and Emulate Urban Climate on AWS Cloud.

Author: Dr. Zhonghua Zheng

CHAPTER

ONE

# ABOUT

## 1.1 Introduction

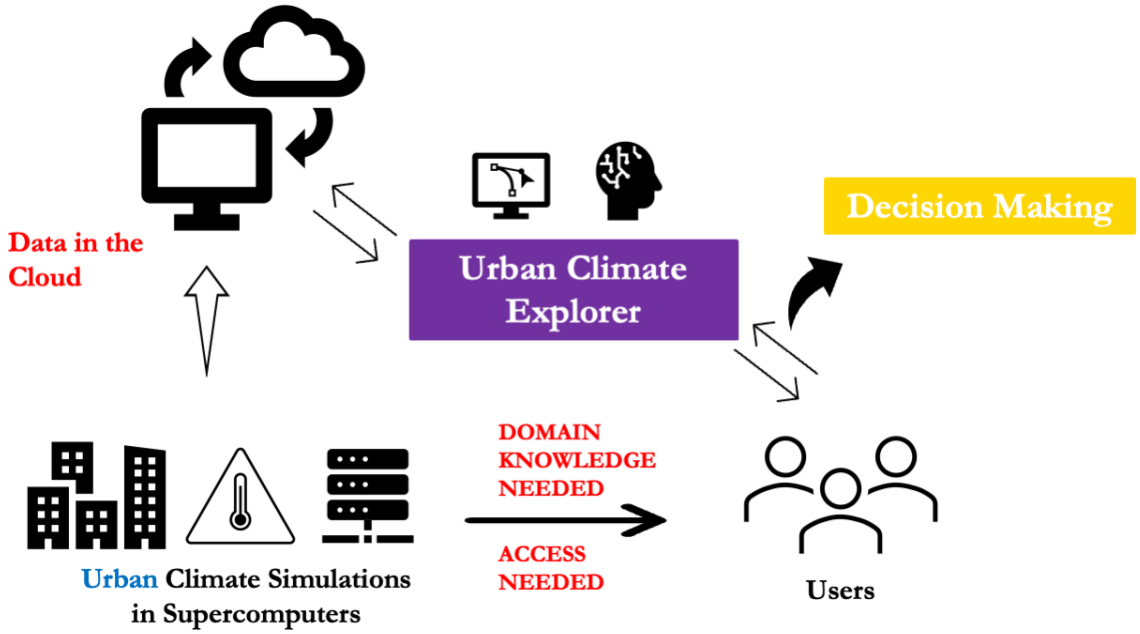
This platform enables free and easy access to the **urban** climate simulations provided by National Center for Atmospheric Research and Amazon Web Services (AWS) via Cloud Computing. By providing the necessary information as the input (e.g., time, latitude, longitude, climate scenarios, etc.), users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

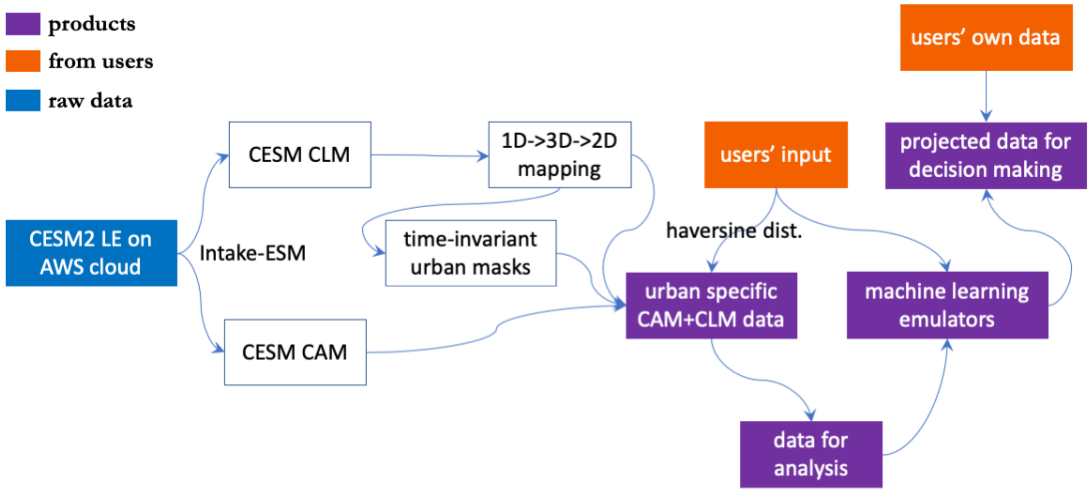

- Zheng, Z., Zhao, L. & Oleson, K.W. Large model structural uncertainty in global projections of urban heat waves. *Nat Commun* **12,** 3736 (2021). https://doi.org/10.1038/s41467-021-24113-9
- Zhao, L., Oleson, K., Bou-Zeid, E. *et al.* Global multi-model projections of local urban climates. *Nat. Clim.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

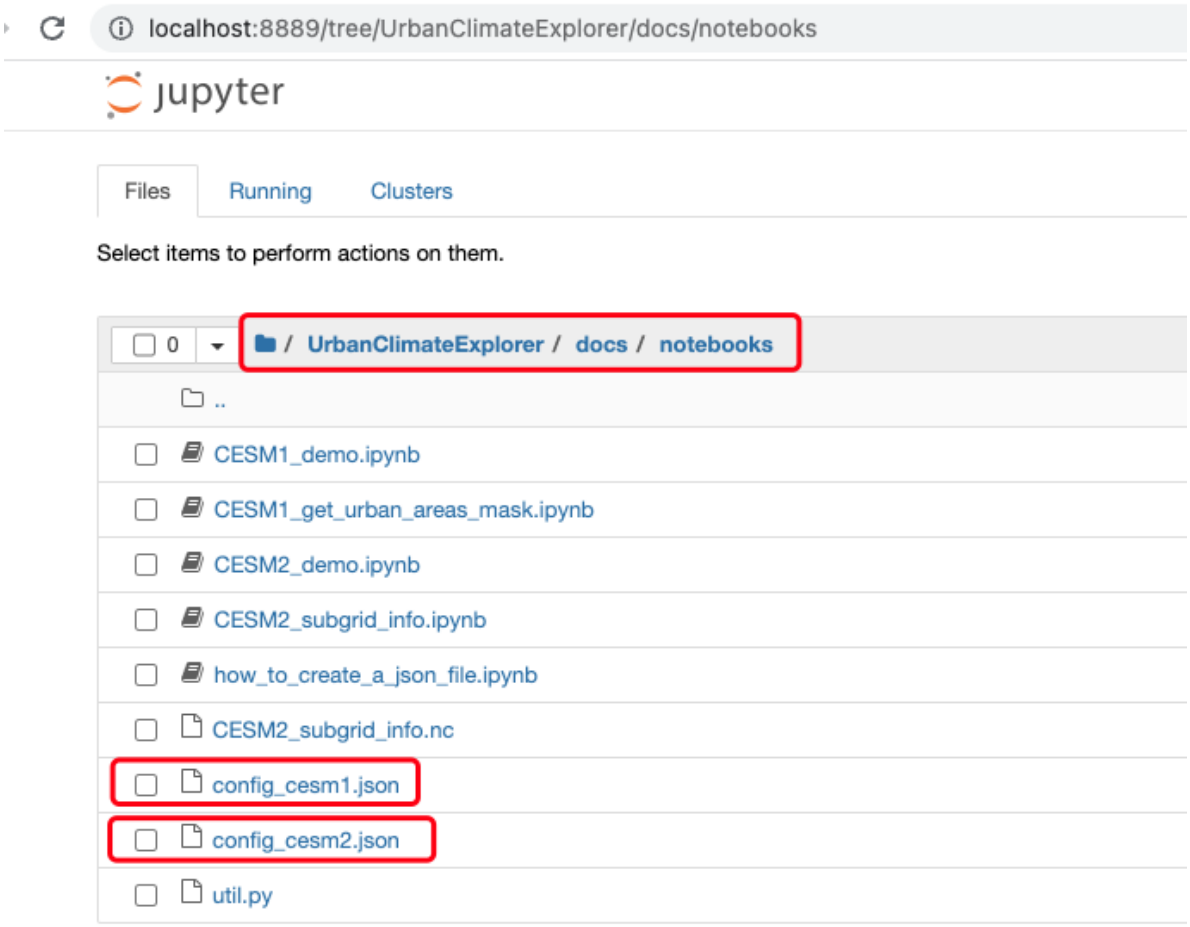
```
#!/bin/bash
source  /glade/work/zhonghua/miniconda3/bin/activate aws_urban
echo "ssh -N -L 8889:`hostname`:8892 $USER@`hostname`.ucar.edu"
jupyter notebook --no-browser --ip=`hostname` --port=8892
```

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`.

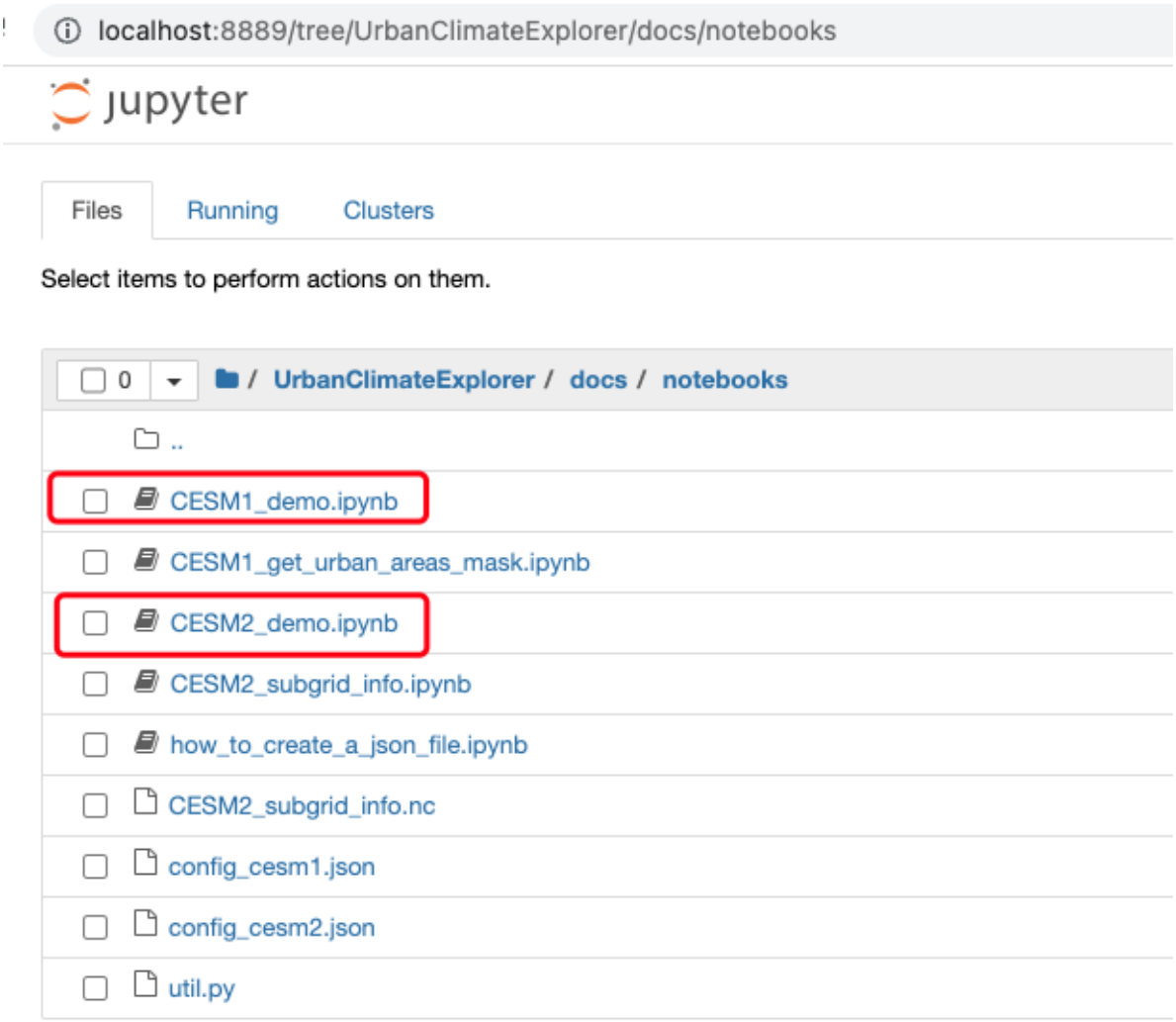



load_JSON

- 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": 40.1164, "lon": -88.2434}
    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 47.5 s, sys: 24.2 s, total: 1min 11s
Wall time: 42.2 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 271.2
  * time        (time) datetime64[ns] 2081-01-02T12:00:00 ... 2100-12-31T12:0...
Data variables:
    TREFHT      (member_id, time) float32 267.5 267.7 275.7 ... 280.6 278.0
    TREFHTMX    (member_id, time) float32 270.9 276.0 283.8 ... 289.3 285.7
    FLNS        (member_id, time) float32 67.13 75.71 53.1 ... 72.42 71.2 62.57
```

(continues on next page)

(continued from previous page)

```
    FSNS        (member_id, time) float32 93.39 89.37 88.2 ... 83.75 89.64 77.89
    PRECSC      (member_id, time) float32 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0
    PRECSL      (member_id, time) float32 3.048e-09 8.637e-10 ... 0.0 0.0
    PRECT       (member_id, time) float32 3.048e-09 8.637e-10 ... 4.164e-10
    QBOT        (member_id, time) float32 0.001285 0.001422 ... 0.004192
    UBOT        (member_id, time) float32 7.931 1.266 -1.619 ... 3.694 -0.1972
    VBOT        (member_id, time) float32 0.4976 2.597 5.257 ... 1.785 0.1336
    TREFMXAV_U  (member_id, time) float32 291.6 271.9 276.9 ... 288.4 289.4
Attributes: (12/14)
    intake_esm_varname:        FLNS\nFSNS\nPRECSC\nPRECSL\nPRECT\nQBOT\nTREFH...
    source:                    CAM
    Conventions:               CF-1.0
    Version:                   $Name$
    revision_Id:               $Id$
    initial_file:              b.e11.B20TRC5CNBDRD.f09_g16.105.cam.i.2006-01-...
    ...                        ...
    nco_openmp_thread_number:  1
    host:                      tcs-f02n07
    topography_file:           /scratch/p/pjk/mudryk/cesm1_1_2_LENS/inputdata...
    title:                     UNSET
    NCO:                       4.4.2
    intake_esm_dataset_key:    atm.RCP85.daily
```

**pandas dataframe**

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

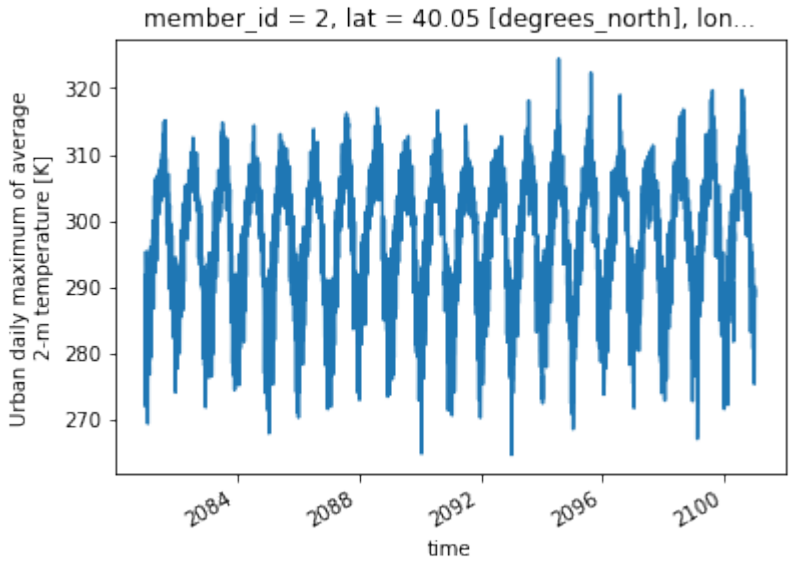
[3]:

| | 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 | 267.530823 | 270.858276 | 67.127922 | 93.389015 | 0.0 | 3.047973e-09 | 3.047973e-09 | 0.001285 | 7.930864 | 0.497645 | 40.052357 | 271.25 | 291.566376 | 3.047973e-09 |
| 1 | 2 | 2081-01-03 12:00:00 | 267.687714 | 276.013184 | 75.707771 | 89.367584 | 0.0 | 8.636923e-10 | 8.637052e-10 | 0.001422 | 1.265992 | 2.597399 | 40.052357 | 271.25 | 271.861420 | 8.636923e-10 |
| 2 | 2 | 2081-01-04 12:00:00 | 275.672028 | 283.799103 | 53.098457 | 88.202408 | 0.0 | 1.331122e-09 | 2.900762e-09 | 0.001883 | -1.618823 | 5.256704 | 40.052357 | 271.25 | 276.851349 | 1.331122e-09 |
| 3 | 2 | 2081-01-05 12:00:00 | 276.890686 | 284.918884 | 38.585438 | 43.366203 | 0.0 | 0.000000e+00 | 1.297357e-09 | 0.004626 | -2.739517 | 5.846395 | 40.052357 | 271.25 | 284.498749 | 0.000000e+00 |
| 4 | 2 | 2081-01-06 12:00:00 | 287.462036 | 291.275452 | 7.098297 | 38.443039 | 0.0 | 1.156371e-14 | 6.182010e-08 | 0.009604 | -0.635833 | 8.809414 | 40.052357 | 271.25 | 285.547607 | 1.156371e-14 |

**data visualization**

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

[4]:
```
[<matplotlib.lines.Line2D at 0x2b4ed40910d0>]
```




## 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)
```

```
LGBMRegressor(colsample_bytree=0.6649148062238498,
              learning_rate=0.17402065726724145, max_bin=255,
              min_child_samples=3, n_estimators=93, num_leaves=15,
              reg_alpha=0.0009765625, reg_lambda=0.0067613624509965,
              verbose=-1)
CPU times: user 3min 4s, sys: 3.09 s, total: 3min 7s
Wall time: 15.5 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")
```

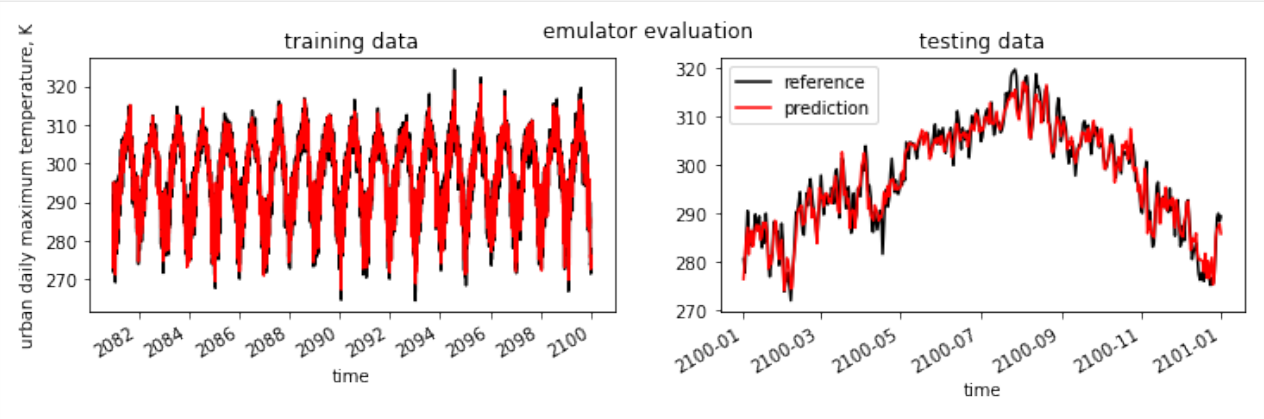
```
model performance using training data:
root mean square error: 1.891482463886013
r2: 0.9646972730399841

model performance using testing data:
root mean square error: 2.4148976103187043
r2: 0.9496869325210374
```

**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
320
310
300
290
280
270
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": 40.1164, "lon": -88.2434}
    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 55.3 s, sys: 32 s, total: 1min 27s
Wall time: 53.7 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 271.2
  * 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 275.0 272.9 273.8 ... 274.1 276.3 281.3
    TREFHTMX   (member_id, time) float32 277.3 274.8 275.4 ... 278.0 283.7 282.8
    FLNS       (member_id, time) float32 59.86 68.45 24.26 ... 90.92 70.91 12.33
    FSNS       (member_id, time) float32 90.96 74.59 38.51 ... 91.1 79.44 22.48
    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 1.517e-10 3.642e-09 ... 0.0 0.0
    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 4.767e-10 3.642e-09 ... 1.786e-09
    TREFMXAV   (member_id, time) float64 277.5 275.5 275.8 ... 278.5 283.8 283.1
Attributes:
    intake_esm_varname:      FLNS\nFSNS\nPRECC\nPRECL\nPRECSC\nPRECSL\nTREFHT...
    Conventions:             CF-1.0
    logname:                 sunseon
    source:                  CAM
    model_doi_url:           https://doi.org/10.5065/D67H1H0V
    time_period_freq:        day_1
    host:                    mom1
    topography_file:         /mnt/lustre/share/CESM/cesm_input/atm/cam/topo/f...
    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 | 275.049347 | 277.329407 | 59.856152 | 90.956940 | 0.0 | 1.516594e-10 | 0.0 | 4.766932e-10 | 40.052356 | 271.25 | 277.536102 | 1.516594e-10 | 4.766932e-10 |
| 1 | r1i1231p1f1 | 2081-01-03 12:00:00 | 272.927429 | 274.842499 | 68.446648 | 74.588417 | 0.0 | 3.641785e-09 | 0.0 | 3.642020e-09 | 40.052356 | 271.25 | | | |
| 2 | r1i1231p1f1 | 2081-01-04 12:00:00 | 273.798920 | 275.402435 | 24.263290 | 38.506813 | 0.0 | 1.956359e-09 | 0.0 | 2.209635e-09 | 40.052356 | 271.25 | | | |
| 3 | r1i1231p1f1 | 2081-01-05 12:00:00 | 277.125549 | 286.210144 | 59.307308 | 86.190071 | 0.0 | 1.833845e-14 | 0.0 | 4.872912e-14 | 40.052356 | 271.25 | | | |
| 4 | r1i1231p1f1 | 2081-01-06 12:00:00 | 280.623657 | 282.968323 | 38.607136 | 17.733625 | 0.0 | 3.573996e-24 | 0.0 | 2.074043e-10 | 40.052356 | 271.25 | | | |

(continues on next page)

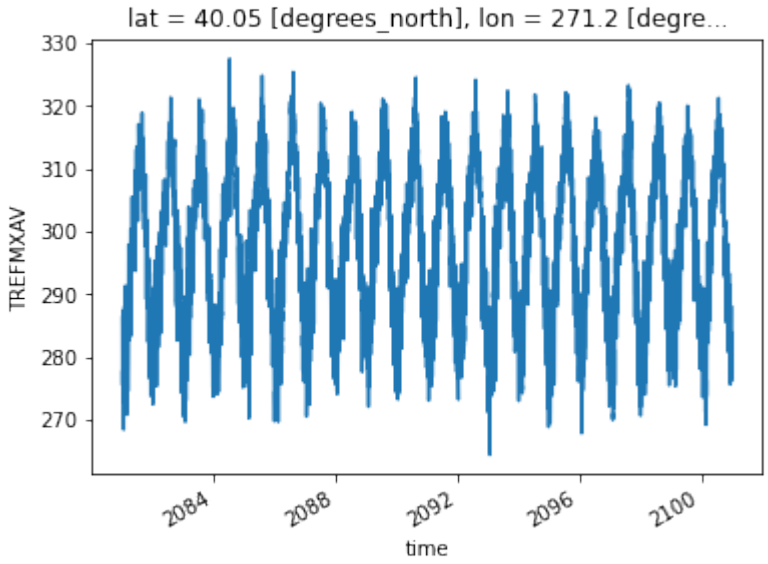
(continued from previous page)

```
1  275.493225  3.641785e-09  3.642020e-09
2  275.750916  1.956359e-09  2.209635e-09
3  286.460663  1.833845e-14  4.872912e-14
4  284.418915  3.573996e-24  2.074043e-10
```

**data visualization**

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

[4]:
```
[<matplotlib.lines.Line2D at 0x2b60524db1c0>]
```




## 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 4s, sys: 4.67 s, total: 3min 8s
Wall time: 15.5 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")
```

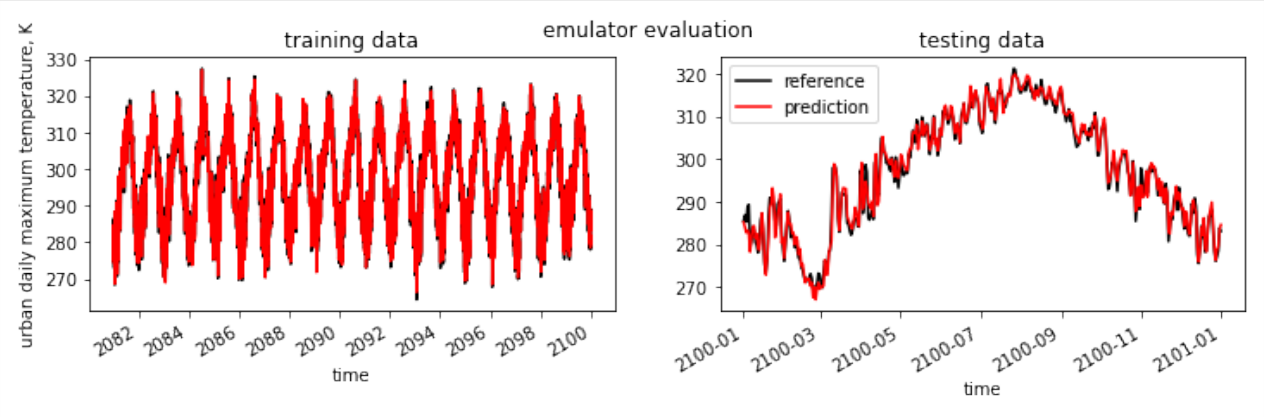
```
model performance using training data:
root mean square error: 1.0634138507297426
r2: 0.9928674646696605

model performance using testing data:
root mean square error: 1.6361120260552937
r2: 0.9852130533126713
```

**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
330
320
310
300
290
280
270
2082
2084
2086
2088
2090
2092
2094
2096
2098
2100
time
reference
prediction
320
310
300
290
280
270
2100-01
2100-03
2100-05
2100-07
2100-09
2100-11
2101-01
time

CHAPTER

SIX

# 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

SEVEN

# 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>
```

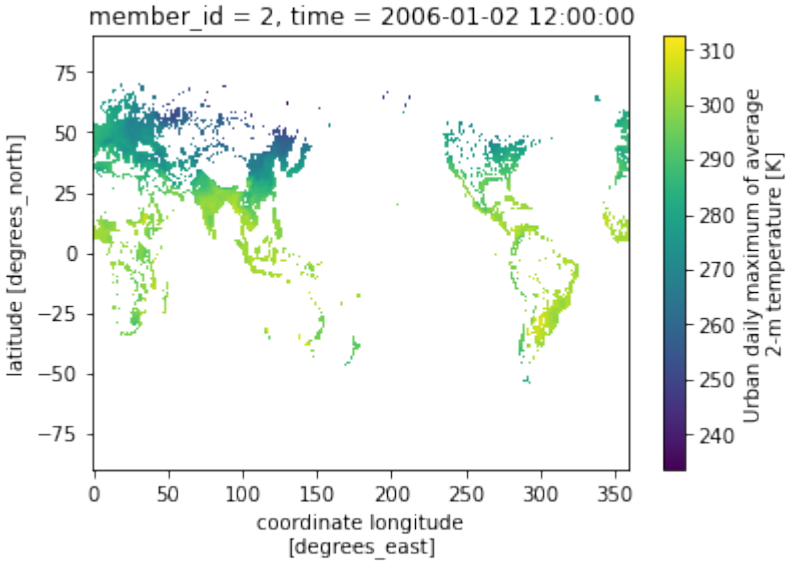
```
<IPython.core.display.HTML object>
```

**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**

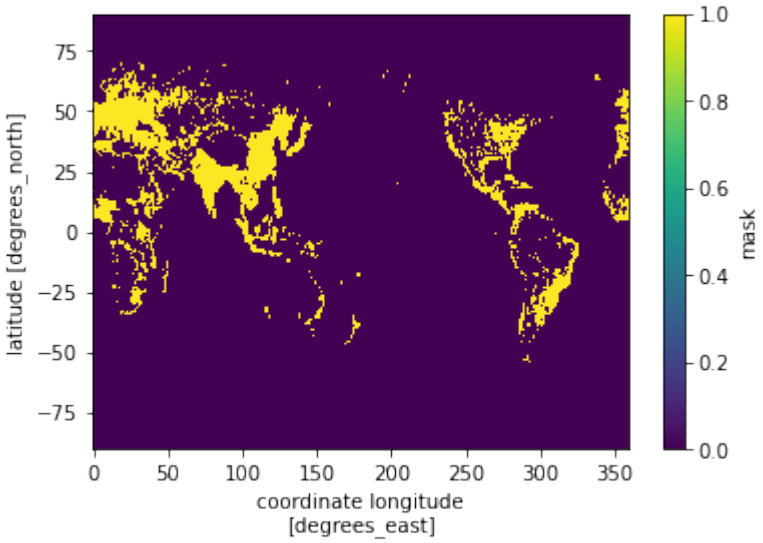
The file is save at current working directory, with a file name "urban_mask.nc"

[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

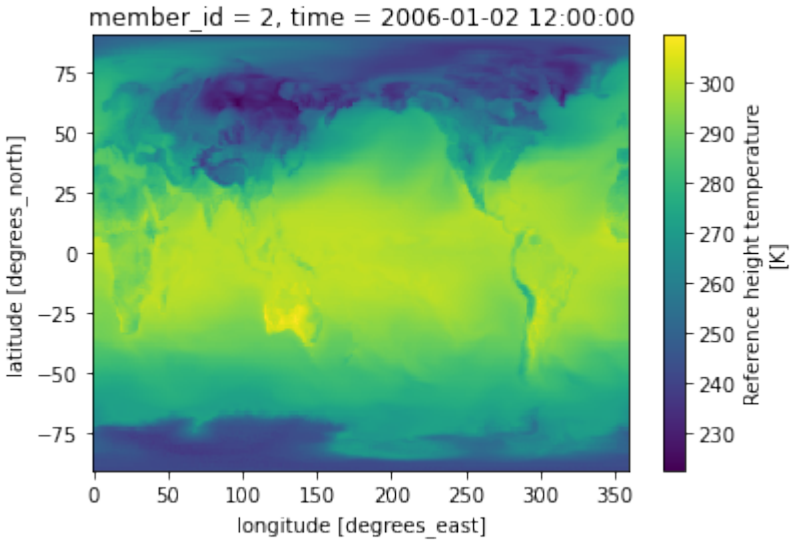
[6]:
```
col_subset = col.search(experiment=experiment, frequency=frequency, variable=cam_
↪variable)
dsets = col_subset.to_dataset_dict(zarr_kwargs={"consolidated": True},
                                   storage_options={"anon": True})['atm.RCP85.daily']
da_cam = dsets.sel(member_id=2, time="2006-01-02")[cam_variable].load()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

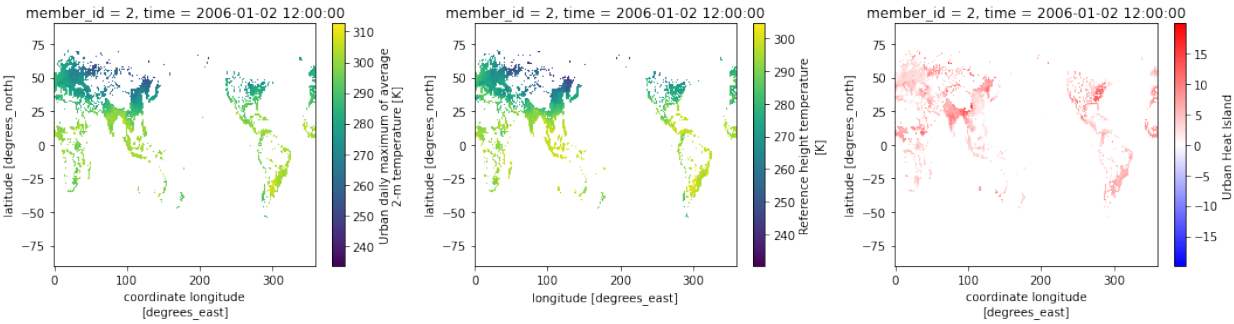
[6]:
```
<matplotlib.collections.QuadMesh at 0x2abf6164fd60>
```




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

# EIGHT

# 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

NINE

# HOW TO ASK FOR HELP?

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

CHAPTER

TEN

# 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.