Expanding GeoCAT's Climatology Resources to Support the Transition from NCL to Python

Andy McKeen[1,2], Anissa Zacharias[1], Katelyn FitzGerald[1]

[1] NSF National Center for Atmospheric Research (NSF NCAR)
[2] Vermont State University Lyndon

NCL to Python

- In 2019, NSF NCAR decided to "Pivot to Python" and put NCL into "Maintenance Mode"
- NCL was becoming difficult to maintain, and Python was free, open-source, and already popular among the scientific community
- The GeoCAT team was created to ensure the transition was as seamless as possible

NCAR Command Language

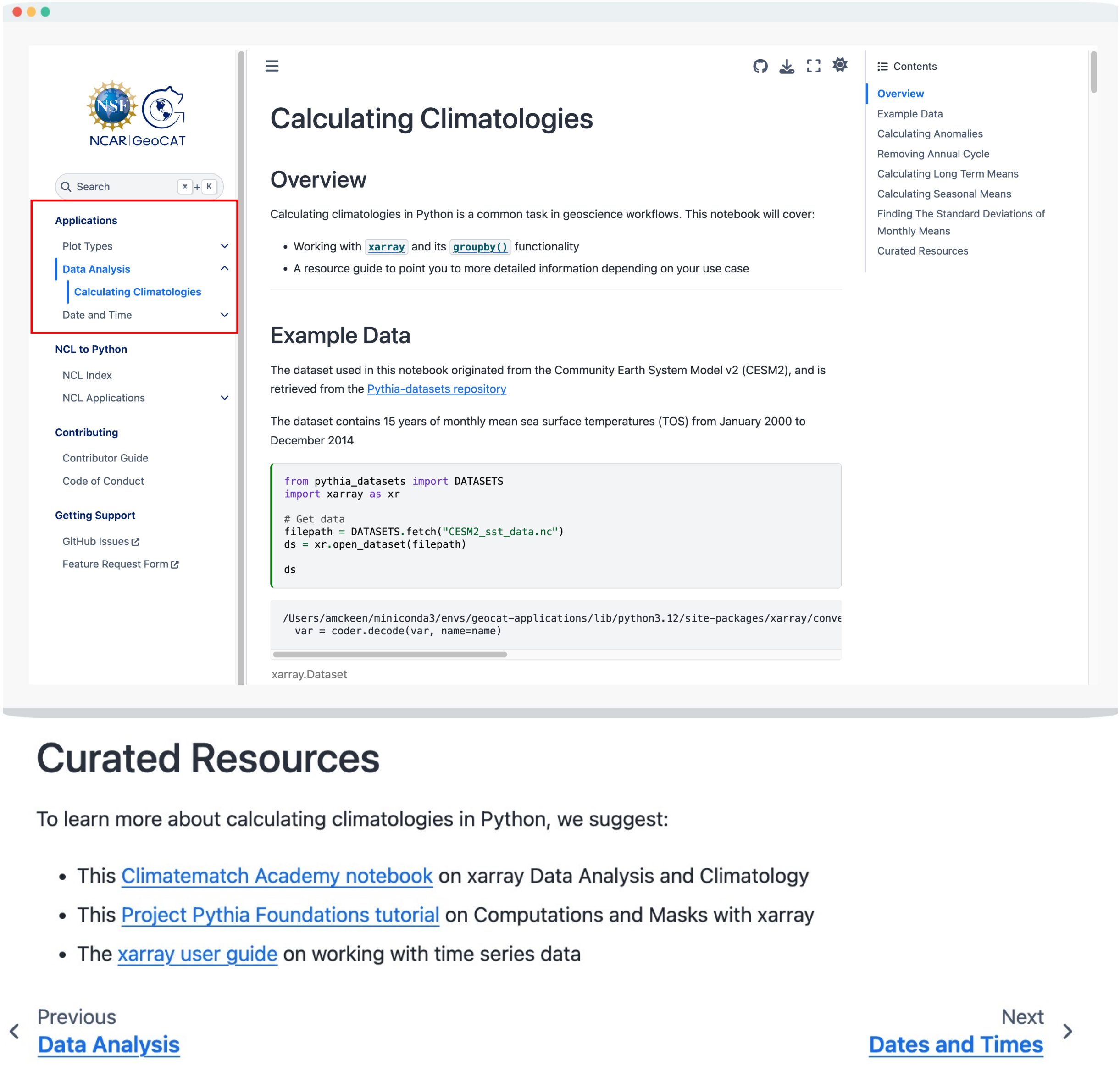
GeoCAT Applications

- Inspired by the NCL Applications page
- Designed to be a quick reference demonstrating capabilities within the scientific Python ecosystem
- Contains both Python-first content, and NCL to Python content

GeoCAT Applications GitHub

GeoCAT Applications Website

Climatology



Calculating Climatologies

Overview

Calculating climatologies in Python is a common task in geoscience workflows. This notebook will cover:

- Working with `xarray` and its `groupby()` functionality
- A resource guide to point you to more detailed information depending on your use case

Example Data

The dataset used in this notebook originated from the Community Earth System Model v2 (CESM2), and is retrieved from the Pythia-datasets repository

The dataset contains 15 years of monthly mean sea surface temperatures (TOS) from January 2000 to December 2014

```
from pythia_datasets import DATASETS
import xarray as xr

# Get data
filepath = DATASETS.fetch("CESM2_sst_data.nc")
ds = xr.open_dataset(filepath)

ds
```

```
/Users/amckeen/miniconda3/envs/geocat-applications/lib/python3.12/site-packages/xarray/conve
  var = coder.decode(var, name=name)
```

xarray.Dataset

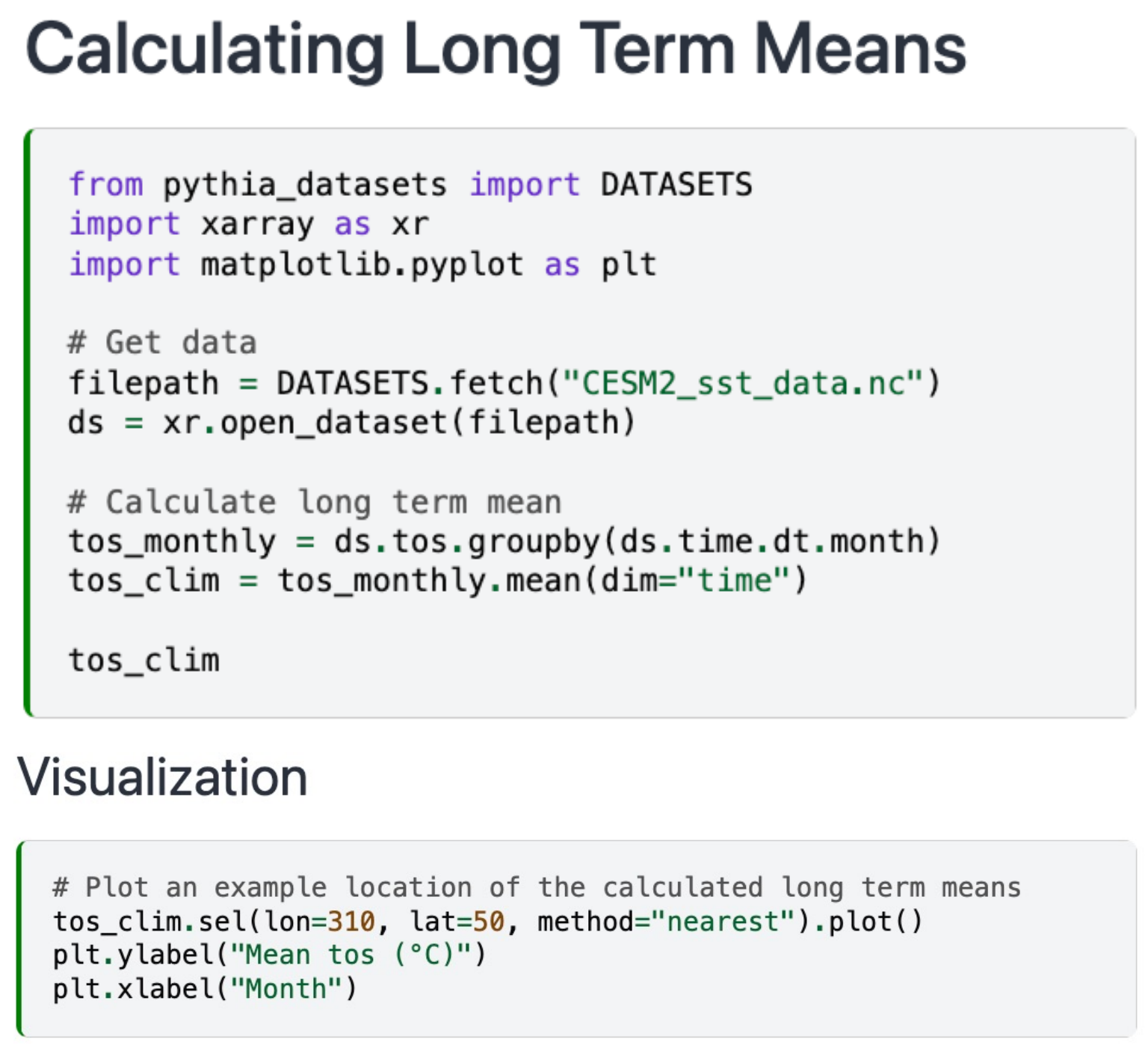
Curated Resources

To learn more about calculating climatologies in Python, we suggest:

- This Climatematch Academy notebook on xarray Data Analysis and Climatology
- This Project Pythia Foundations tutorial on Computations and Masks with xarray
- The xarray user guide on working with time series data

Previous
Data Analysis

Next
Dates and Times

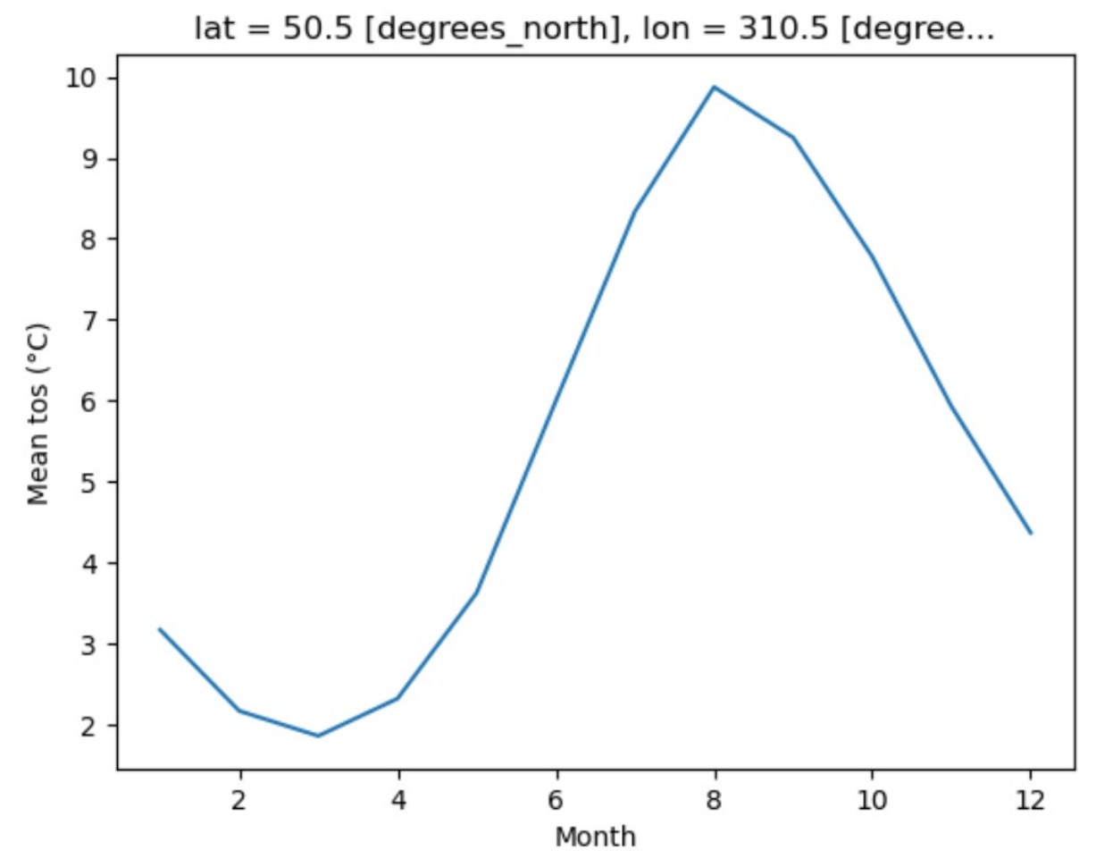


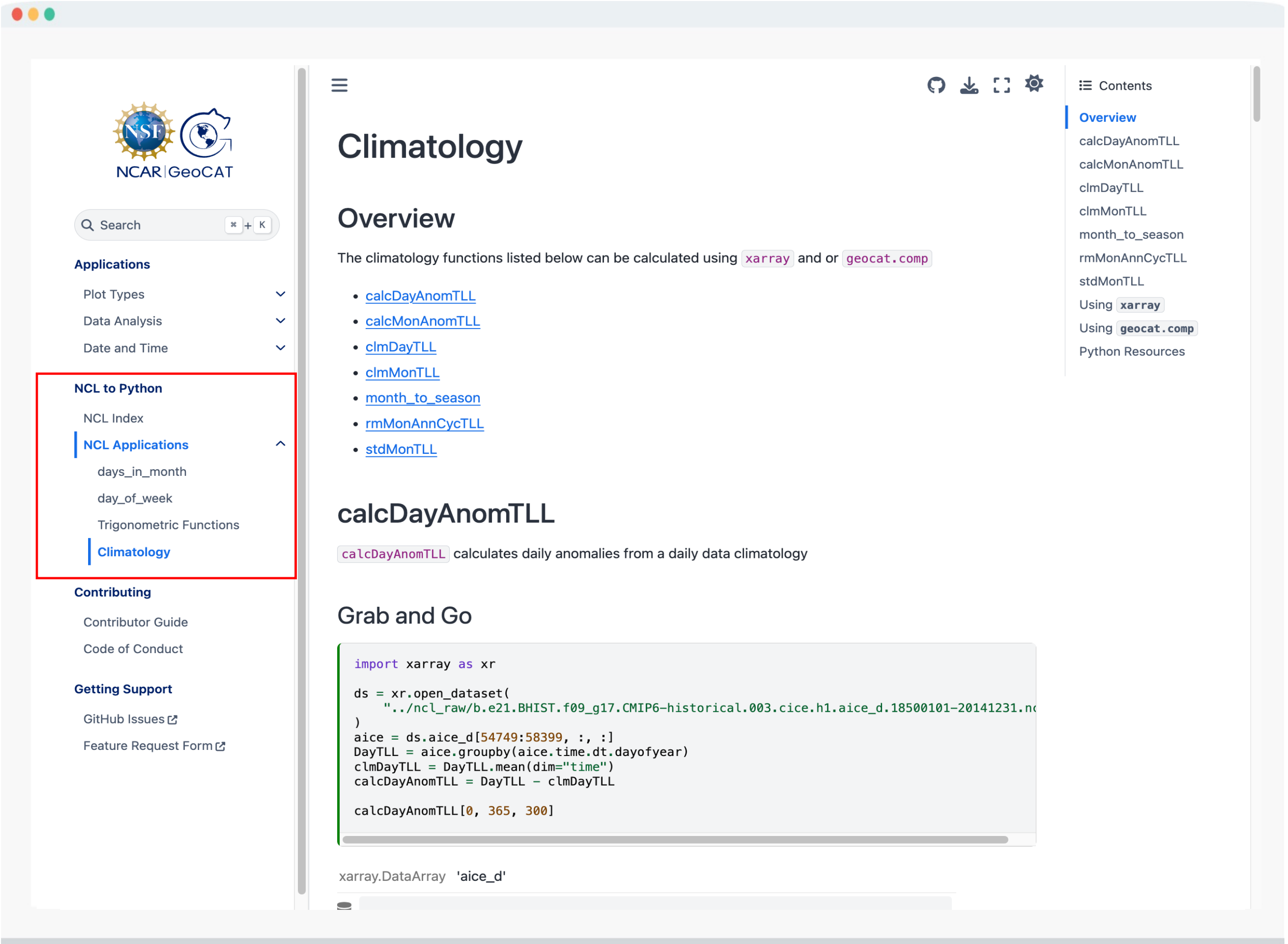
Calculating Long Term Means

```
from pythia_datasets import DATASETS
import xarray as xr
import matplotlib.pyplot as plt

# Get data
filepath = DATASETS.fetch("CESM2_sst_data.nc")
ds = xr.open_dataset(filepath)

# Calculate long term mean
tos_monthly = ds.tos.groupby(ds.time.dt.month)
tos_clim = tos_monthly.mean(dim="time")

tos_clim
```

Visualization

```
# Plot an example location of the calculated long term means
tos_clim.sel(lon=310, lat=50, method="nearest").plot()
plt.ylabel("Mean tos (°C)")
plt.xlabel("Month")
```





Climatology

Overview

The climatology functions listed below can be calculated using `xarray` and or `geocat.comp`

- calcDayAnomTLL
- calcMonAnomTLL
- clmDayTLL
- clmMonTLL
- month_to_season
- rmMonAnnCycTLL
- stdMonTLL

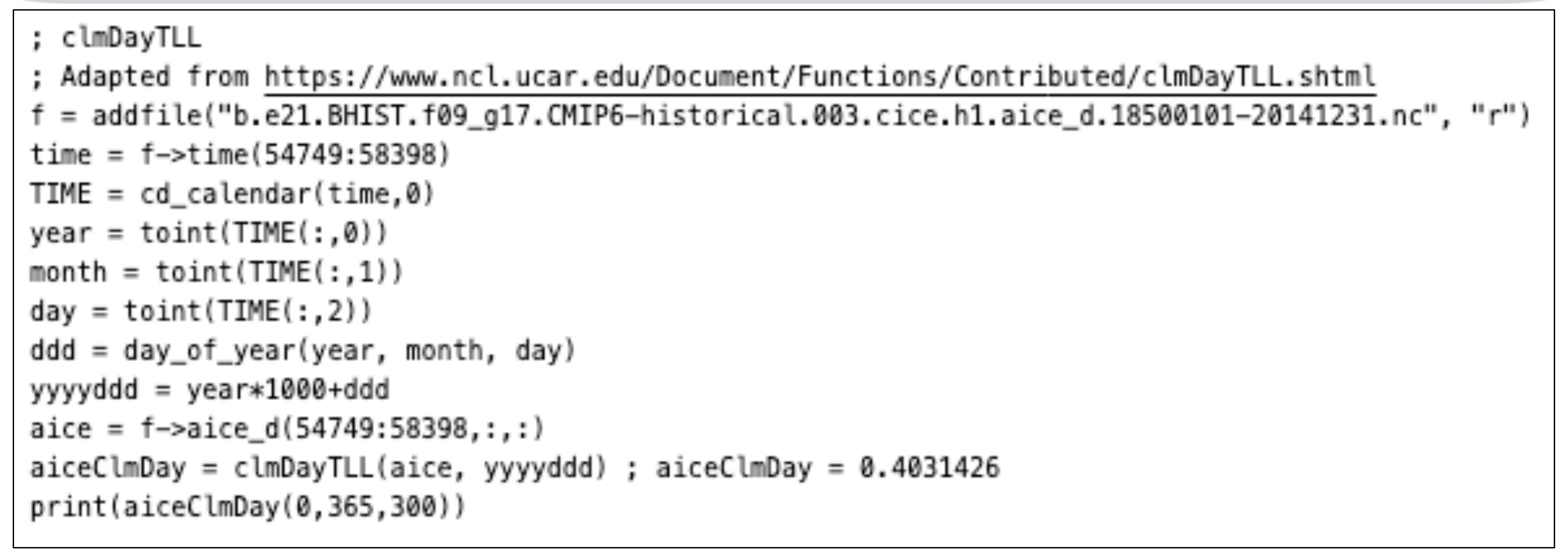
calcDayAnomTLL

`calcDayAnomTLL` calculates daily anomalies from a daily data climatology

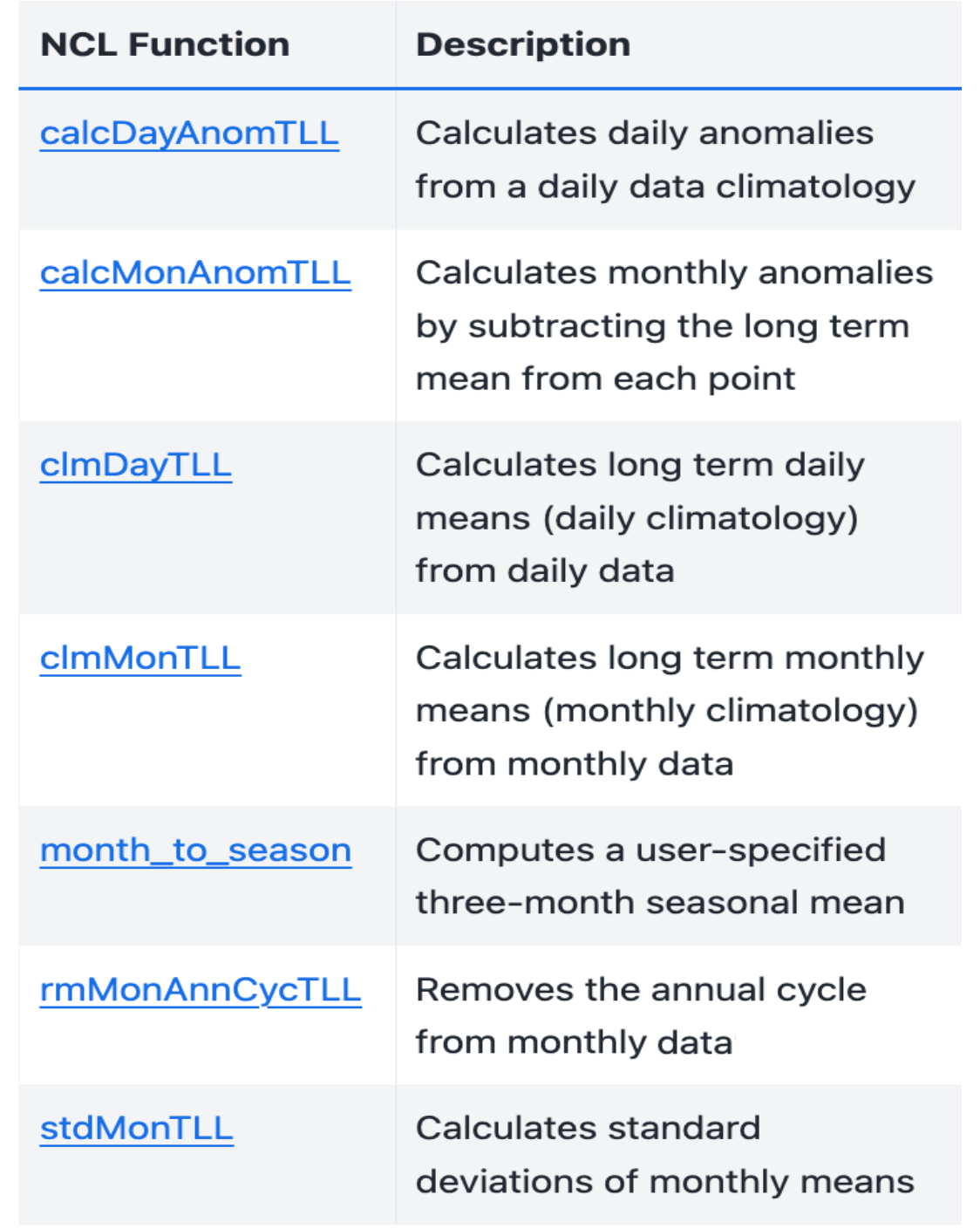
Grab and Go

```
import xarray as xr

ds = xr.open_dataset(
    "../ncl_raw/b.e21.BHIST.f09_g17.CMIP6-historical.003.cice.h1.aice_d.18500101-20141231.nc
)
aice = ds.aice_d[54749:58399, :, :]
DayTLL = aice.groupby(aice.time.dt.dayofyear)
clmDayTLL = DayTLL.mean(dim="time")
calcDayAnomTLL = DayTLL - clmDayTLL

calcDayAnomTLL[0, 365, 300]
```

xarray.DataArray 'aice_d'

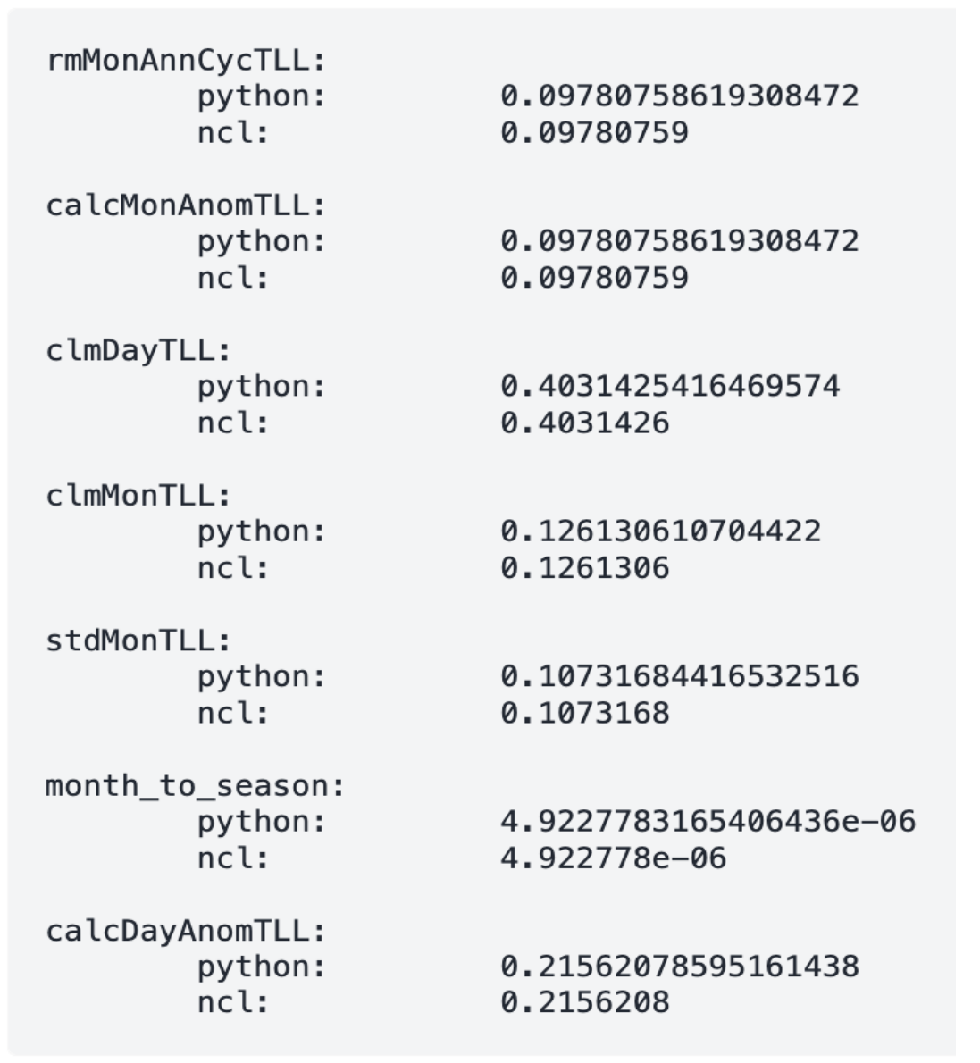


```
; clmDayTLL
; Adapted from https://www.ncl.ucar.edu/Document/Functions/Contributed/clmDayTLL.shtml
f = addfile("b.e21.BHIST.f09_g17.CMIP6-historical.003.cice.h1.aice_d.18500101-20141231.nc", "r")
time = f->time(54749:58398)
TIME = cd_calendar(time,0)
year = toint(TIME(:,0))
month = toint(TIME(:,1))
day = toint(TIME(:,2))
ddd = day_of_year(year, month, day)
yyyyddd = year*1000+ddd
aice = f->aice_d(54749:58398,:,:)
aiceClmDay = clmDayTLL(aice, yyyyddd) ; aiceClmDay = 0.4031426
print(aiceClmDay(0,365,300))
```



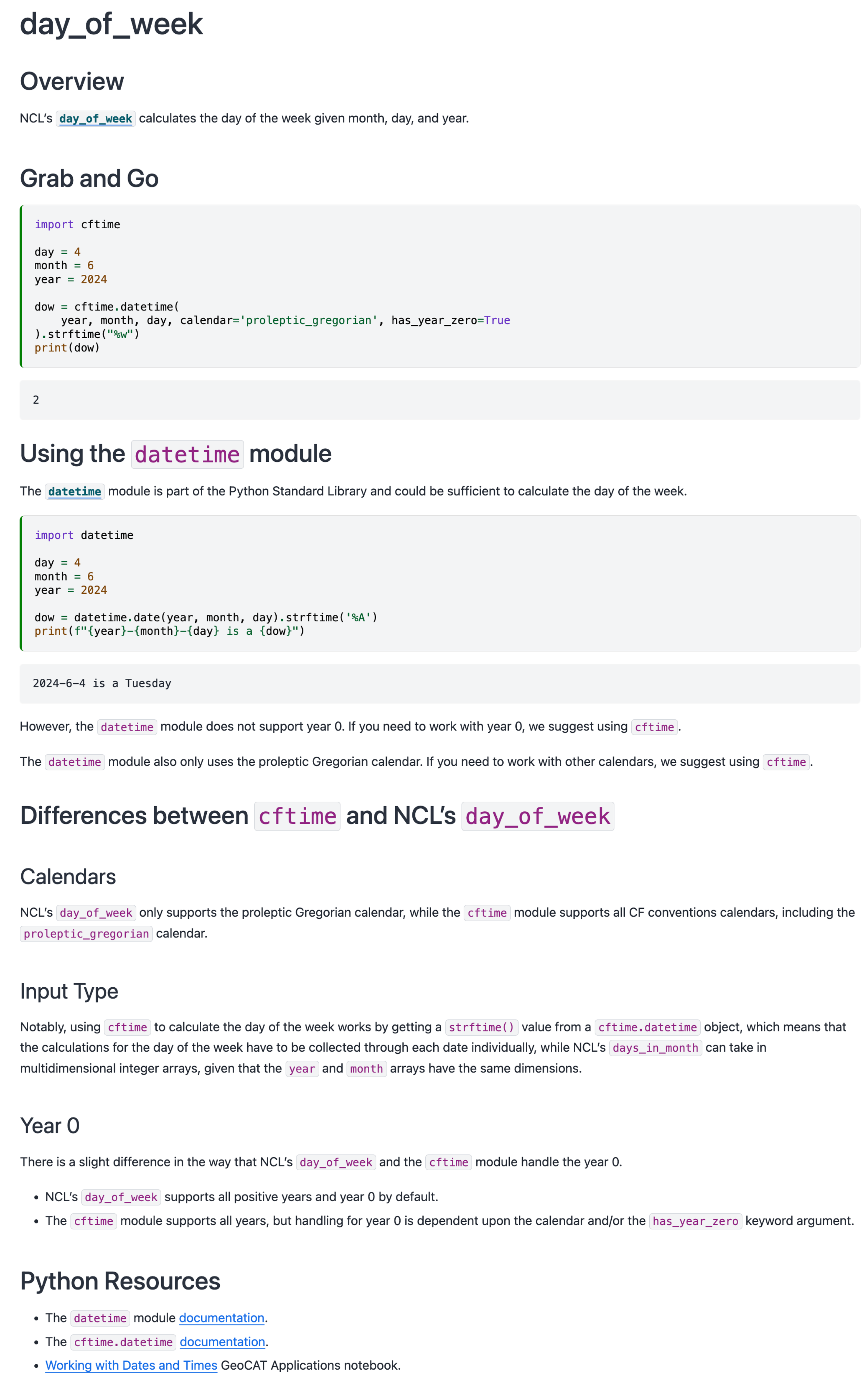
NCL Function	Description
calcDayAnomTLL	Calculates daily anomalies from a daily data climatology
calcMonAnomTLL	Calculates monthly anomalies by subtracting the long term mean from each point
clmDayTLL	Calculates long term daily means (daily climatology) from daily data
clmMonTLL	Calculates long term monthly means (monthly climatology) from monthly data
month_to_season	Computes a user-specified three-month seasonal mean
rmMonAnnCycTLL	Removes the annual cycle from monthly data
stdMonTLL	Calculates standard deviations of monthly means

Comparison



```
rmMonAnnCycTLL:
        python:         0.09780758619308472
        ncl:            0.09780759

calcMonAnomTLL:
        python:         0.09780758619308472
        ncl:            0.09780759

clmDayTLL:
        python:         0.4031425416469574
        ncl:            0.4031426

clmMonTLL:
        python:         0.126130610704422
        ncl:            0.1261306

stdMonTLL:
        python:         0.10731684416532516
        ncl:            0.1073168

month_to_season:
        python:         4.9227783165406436e-06
        ncl:            4.922778e-06

calcDayAnomTLL:
        python:         0.21562078595161438
        ncl:            0.2156208
```

Day of Week



day_of_week

Overview

NCL's `day_of_week` calculates the day of the week given month, day, and year.

Grab and Go

```
import cftime

day = 4
month = 6
year = 2024

dow = cftime.datetime(
    year, month, day, calendar='proleptic_gregorian', has_year_zero=True
).strftime("%w")
print(dow)
```

```
2
```

Using the `datetime` module

The `datetime` module is part of the Python Standard Library and could be sufficient to calculate the day of the week.

```
import datetime

day = 4
month = 6
year = 2024

dow = datetime.date(year, month, day).strftime('%A')
print(f"{year}-{month}-{day} is a {dow}")
```

```
2024-6-4 is a Tuesday
```

However, the `datetime` module does not support year 0. If you need to work with year 0, we suggest using `cftime`.

The `datetime` module also only uses the proleptic Gregorian calendar. If you need to work with other calendars, we suggest using `cftime`.

Differences between `cftime` and NCL's `day_of_week`

Calendars

NCL's `day_of_week` only supports the proleptic Gregorian calendar, while the `cftime` module supports all CF conventions calendars, including the `proleptic_gregorian` calendar.

Input Type

Notably, using `cftime` to calculate the day of the week works by getting a `strftime()` value from a `cftime.datetime` object, which means that the calculations for the day of the week have to be collected through each date individually, while NCL's `days_in_month` can take in multidimensional integer arrays, given that the `year` and `month` arrays have the same dimensions.

Year 0

There is a slight difference in the way that NCL's `day_of_week` and the `cftime` module handle the year 0.

- NCL's `day_of_week` supports all positive years and year 0 by default.
- The `cftime` module supports all years, but handling for year 0 is dependent upon the calendar and/or the `has_year_zero` keyword argument.

Python Resources

- The `datetime` module documentation.
- The `cftime.datetime` documentation.
- Working with Dates and Times GeoCAT Applications notebook.

Future Work

- Add remaining Climatology functions and conduct more difference testing
- Continue to grow the GeoCAT-applications resource