Average for `NaN` included in the data

2024-06-10 Jiwoo Lee (LLNL)

In [ ]:
```
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt
```

Toy code testing

Prepare dummy sample data

In [ ]:
```
# Create a sample DataArray with random NaNs
data = xr.DataArray(np.random.rand(4, 5, 6), dims=("lat", "lon", "time"))
data = data.where(np.random.rand(4, 5, 6) > 0.1)  # Introduce some NaNs
```

In [ ]:
```
data
```

Out[ ]: xarray.DataArray (lat: 4, lon: 5, time: 6)

```
array([[[0.45803863, 0.23684102, 0.34250896, 0.46851904, 0.74736284,
         nan],
        [0.91231666, 0.46362668, 0.45578235, 0.32637482, 0.89301161,
         0.26132585],
        [0.53180638, 0.88712177, 0.47441006, 0.04914012,        nan,
         0.58453821],
        [0.31772218, 0.93996566, 0.84132873,        nan, 0.10666629,
         0.36932333],
        [0.90309314, 0.31648537, 0.10643295, 0.56701448, 0.90206499,
         nan]],

       [[0.37470047, 0.95140686, 0.32902223, 0.65205821, 0.49987217,
         0.46269618],
        [0.88363144, 0.09417366, 0.5883775 , 0.37499195, 0.37479074,
         0.64025816],
        [       nan, 0.0878633 , 0.93069458, 0.74942225, 0.78340806,
         0.34687362],
        [0.7268889 , 0.30751867,        nan,        nan, 0.23500597,
         0.85178077],
        [0.27152839, 0.90622177, 0.9444546 , 0.55957217, 0.42052352,
...
         0.67300538],
        [0.15688618, 0.27111021, 0.84824185, 0.29270785, 0.08328953,
         0.02357834],
        [0.16082658, 0.13608469, 0.78773745,        nan, 0.66215106,
         0.87514719],
        [0.820858  , 0.64385301,        nan, 0.59718858, 0.67275942,
         nan],
        [0.36662626, 0.25401749, 0.14793267, 0.59964198, 0.61931574,
         0.38846455]],

       [[0.25088388, 0.41945115, 0.92853157, 0.11349054, 0.7327055 ,
         0.38749216],
        [0.08084686, 0.58635272, 0.62535689, 0.41316922, 0.33459792,
         0.09426413],
        [0.58640253,        nan, 0.87792995, 0.3218112 , 0.56010791,
         0.75441002],
        [0.29657213, 0.74090224,        nan, 0.53860451, 0.29583759,
         0.9389709 ],
        [0.9549351 , 0.36526909, 0.34965454, 0.54906987, 0.03803496,
         0.84197102]]])
```

▶ Coordinates: (0)

▶ Indexes: (0)

▶ Attributes: (0)

In [ ]:
```
# Count NaNs along each of either spatial and time dimension
nan_count_lat_lon = data.isnull().sum(dim=['lat', 'lon'])
nan_count_time = data.isnull().sum(dim='time')
```

In [ ]:
```
nan_count_lat_lon
```

Out[ ]: xarray.DataArray (time: 6)

array([1, 1, 4, 3, 1, 3])

▶ Coordinates: (0)

▶ Indexes: (0)

▶ Attributes: (0)

In [ ]:
```
nan_count_time
```

Out[ ]: xarray.DataArray (lat: 4, lon: 5)

array([[1, 0, 1, 1, 1],
[0, 0, 1, 2, 0],
[1, 0, 1, 2, 0],
[0, 0, 1, 1, 0]])

▶ Coordinates: (0)

▶ Indexes: (0)

▶ Attributes: (0)

Spatial average

To achieve the spatial average along the lat and lon dimensions for each time step, while checking the number of NaN values and calculating the average only if the number of NaNs is less than 20% of the total data in that time step, you can follow these steps:

1. Count the number of NaN values for each time step along the lat and lon dimensions.
2. Calculate the total number of elements in the lat and lon dimensions for each time step.
3. Check if the number of NaNs is less than 20% of the total elements.
4. Calculate the average if the condition is met; otherwise, set the average as NaN.

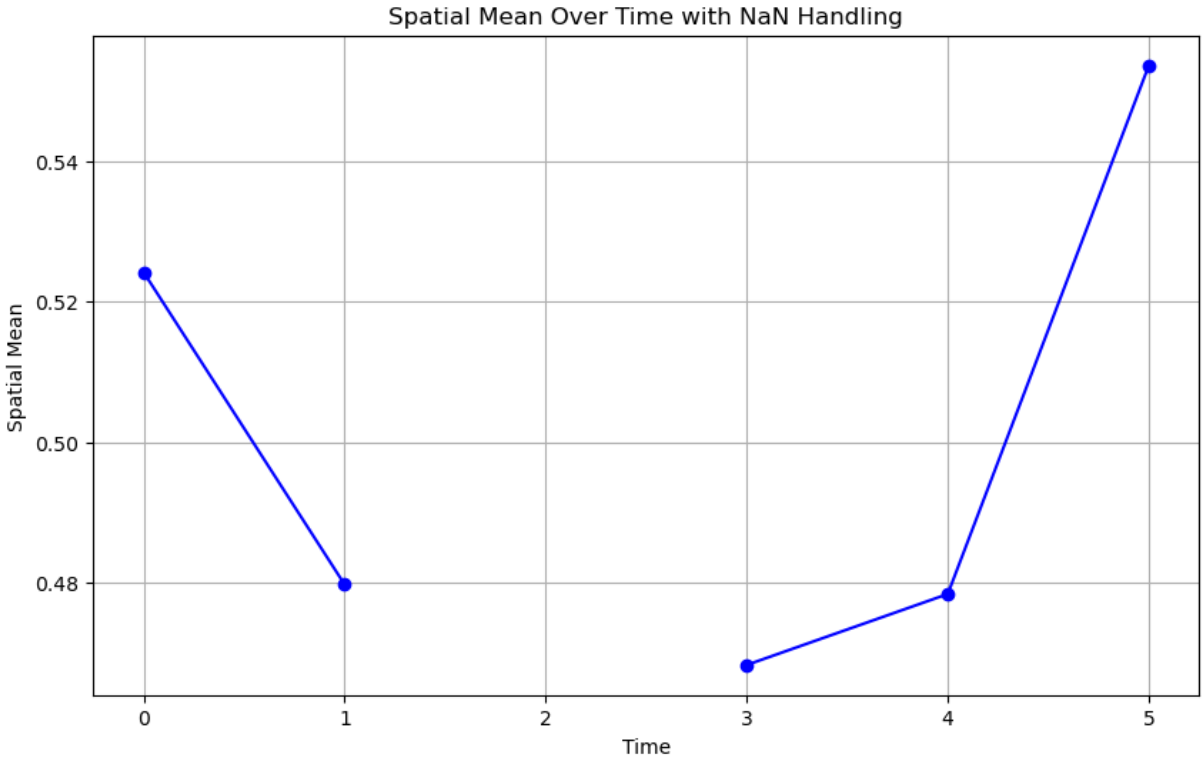
In [ ]:
```
# Dimensions
lat_dim = 'lat'
lon_dim = 'lon'
time_dim = 'time'
```

In [ ]:
```
nan_threshold_fraction = 0.2
```

In [ ]:
```
# Total number of elements along lat and lon dimensions for each time step
total_elements = data.sizes[lat_dim] * data.sizes[lon_dim]

# Count NaNs over the 'lat' and 'lon' dimensions for each time step
nan_count = data.isnull().sum(dim=[lat_dim, lon_dim])

# Determine if the number of NaNs is less than certain fraction of the total
nan_threshold_count = nan_threshold_fraction * total_elements
valid_data_mask = nan_count < nan_threshold_count

# Calculate the mean along the 'lat' and 'lon' dimensions, ignoring NaNs
spatial_mean = data.mean(dim=[lat_dim, lon_dim], skipna=True)

# Set the average to NaN if the number of NaNs exceeds the threshold
spatial_mean = spatial_mean.where(valid_data_mask, other=np.nan)

print("Spatial mean for each time step with NaN handling:\n", spatial_mean)
```

```
Spatial mean for each time step with NaN handling:
 <xarray.DataArray (time: 6)> Size: 48B
array([0.52420493, 0.47973675,        nan, 0.46828287, 0.47838331,
       0.55363154])
Dimensions without coordinates: time
```

In [ ]:
```
# Plotting the spatial mean as a line plot with circle markers
plt.figure(figsize=(10, 6))
plt.plot(spatial_mean.time, spatial_mean, marker='o', linestyle='-', color='
plt.xlabel('Time')
plt.ylabel('Spatial Mean')
plt.title('Spatial Mean Over Time with NaN Handling')
plt.grid(True)
plt.show()
```



Temporal average

To calculate the temporal average only when the number of NaN values is less than 20% for each point in the spatial dimensions (lat and lon), you can follow a similar approach as for the spatial averaging.

Here are the steps:

1. Count the number of NaN values for each point along the time dimension.
2. Calculate the total number of elements along the time dimension.
3. Check if the number of NaNs is less than 20% of the total elements for each point in the lat and lon dimensions.
4. Calculate the average if the condition is met; otherwise, set the average as NaN.

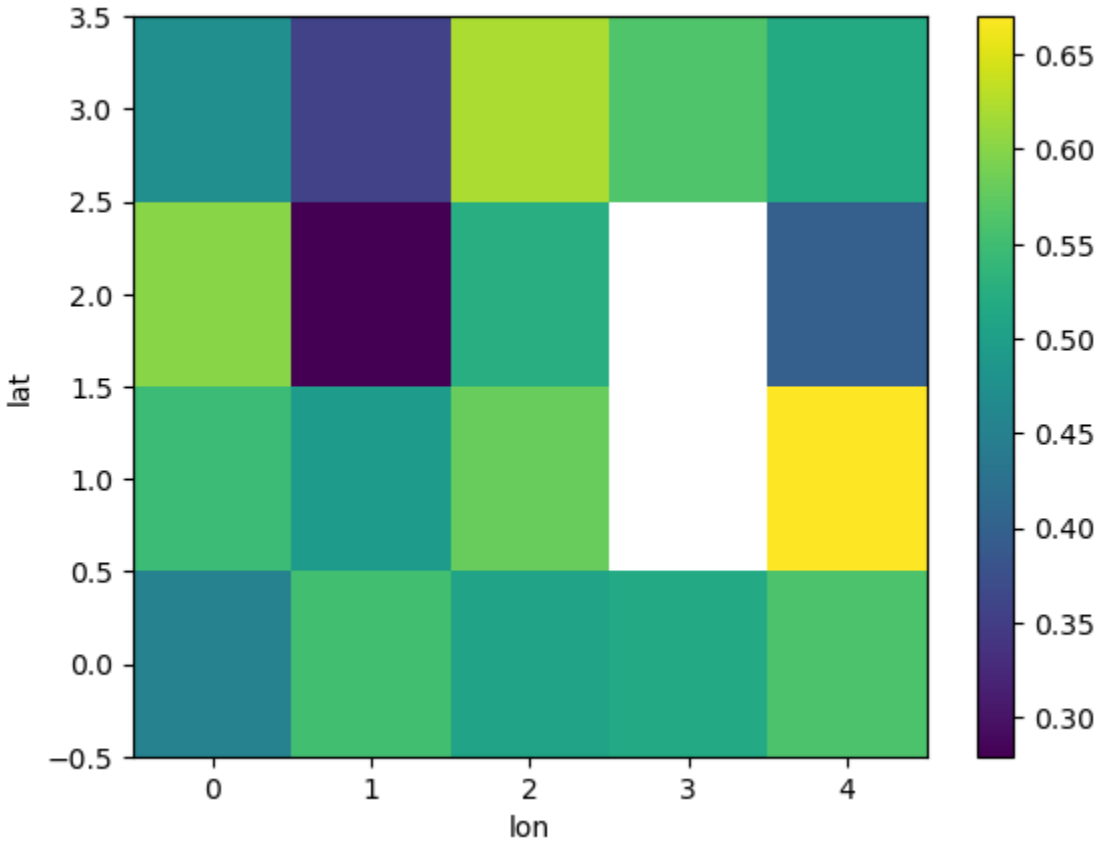
In [ ]:
```
# Total number of elements along the time dimension
total_elements = data.sizes[time_dim]

# Count NaNs over the 'time' dimension for each (lat, lon) point
nan_count = data.isnull().sum(dim=time_dim)

# Determine if the number of NaNs is less than certain fraction of the total
nan_threshold_count = nan_threshold_fraction * total_elements
valid_data_mask = nan_count < nan_threshold_count

# Calculate the mean along the 'time' dimension, ignoring NaNs
temporal_mean = data.mean(dim=time_dim, skipna=True)

# Set the average to NaN if the number of NaNs exceeds the threshold
temporal_mean = temporal_mean.where(valid_data_mask, other=np.nan)
```

```
print("Temporal mean for each spatial point with NaN handling:\n", temporal_
```

```
Temporal mean for each spatial point with NaN handling:
 <xarray.DataArray (lat: 4, lon: 5)> Size: 160B
array([[0.4506541 , 0.55207299, 0.50540331, 0.51500124, 0.55901819],
       [0.54495935, 0.4927039 , 0.57965236,        nan, 0.66998948],
       [0.60017546, 0.27930233, 0.5243894 ,        nan, 0.39599978],
       [0.47209247, 0.35576462, 0.62013232, 0.56217747, 0.5164891 ]])
Dimensions without coordinates: lat, lon
```

In [ ]:
```
temporal_mean.plot()
```

Out[ ]:
```
<matplotlib.collections.QuadMesh at 0x14a0deed0>
```



Real world data

Prepare data

Load data and add `NaN` values

In [ ]:
```
import xcdat as xc
```

In [ ]:
```
inputfile = "/Users/lee1043/Documents/Research/git/pcmdi_metrics_20230620_pc
data_var = "ts"

ds = xc.open_dataset(inputfile).isel(time=slice(0,12*5))
```

In [ ]:
```
ds
```

Out[ ]: xarray.Dataset

▸ Dimensions: (**time**: 60, bnds: 2, **lat**: 145, **lon**: 192)

▾ Coordinates:

lat	(lat)	float64	-90.0 -88.75 -87.5 ... 88.75 90.0
lon	(lon)	float64	0.0 1.875 3.75 ... 356.2 358.1
time	(time)	object	1850-01-16 12:00:00 ... 1854-12...

▾ Data variables:

time_bnds	(time, bnds)	object	...
lat_bnds	(lat, bnds)	float64	...
lon_bnds	(lon, bnds)	float64	...
ts	(time, lat, lon)	float32	...

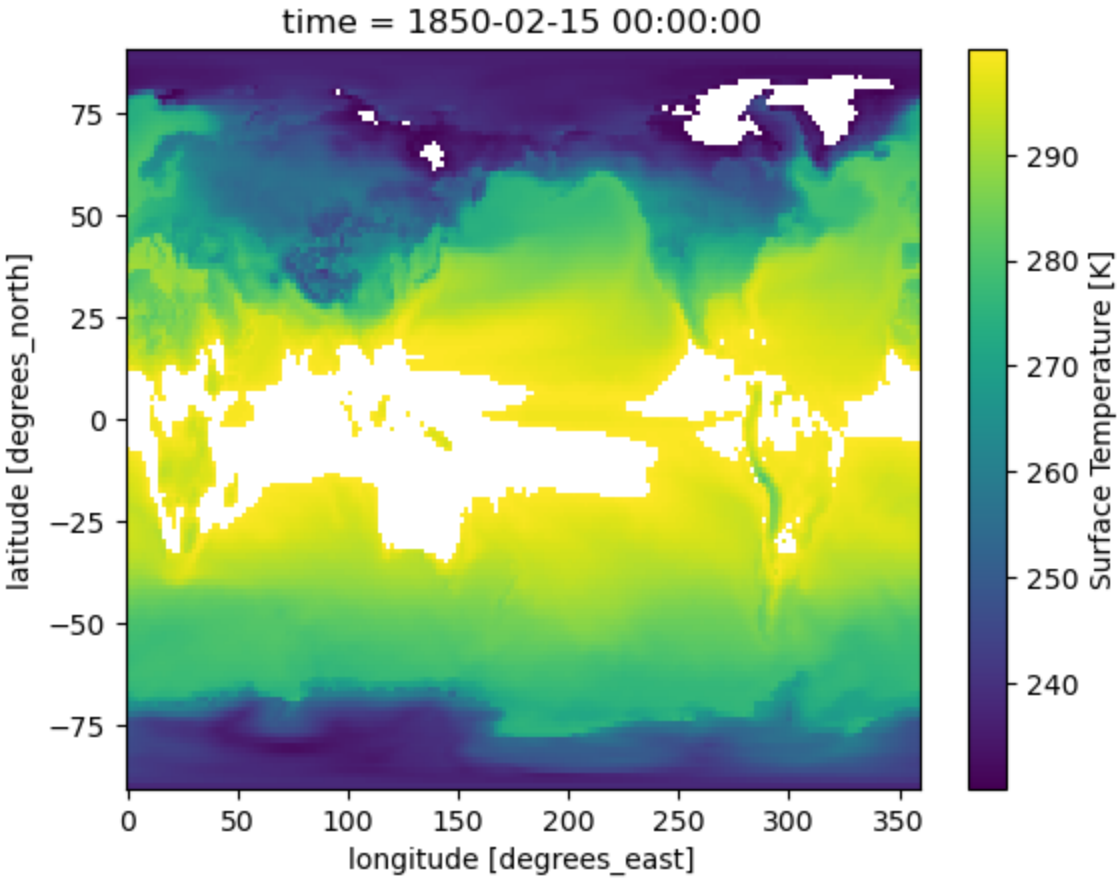
▸ Indexes: (3)

▸ Attributes: (28)

In [ ]:
```
# Inject some random NaNs into the dataArray
ds[data_var].values = ds[data_var].where((ds[data_var] < 300) & (ds[data_var
```

In [ ]:
```
ds[data_var].isel(time=1).plot()
```

Out[ ]:
```
<matplotlib.collections.QuadMesh at 0x155a78310>
```



Spatial average

In [ ]:

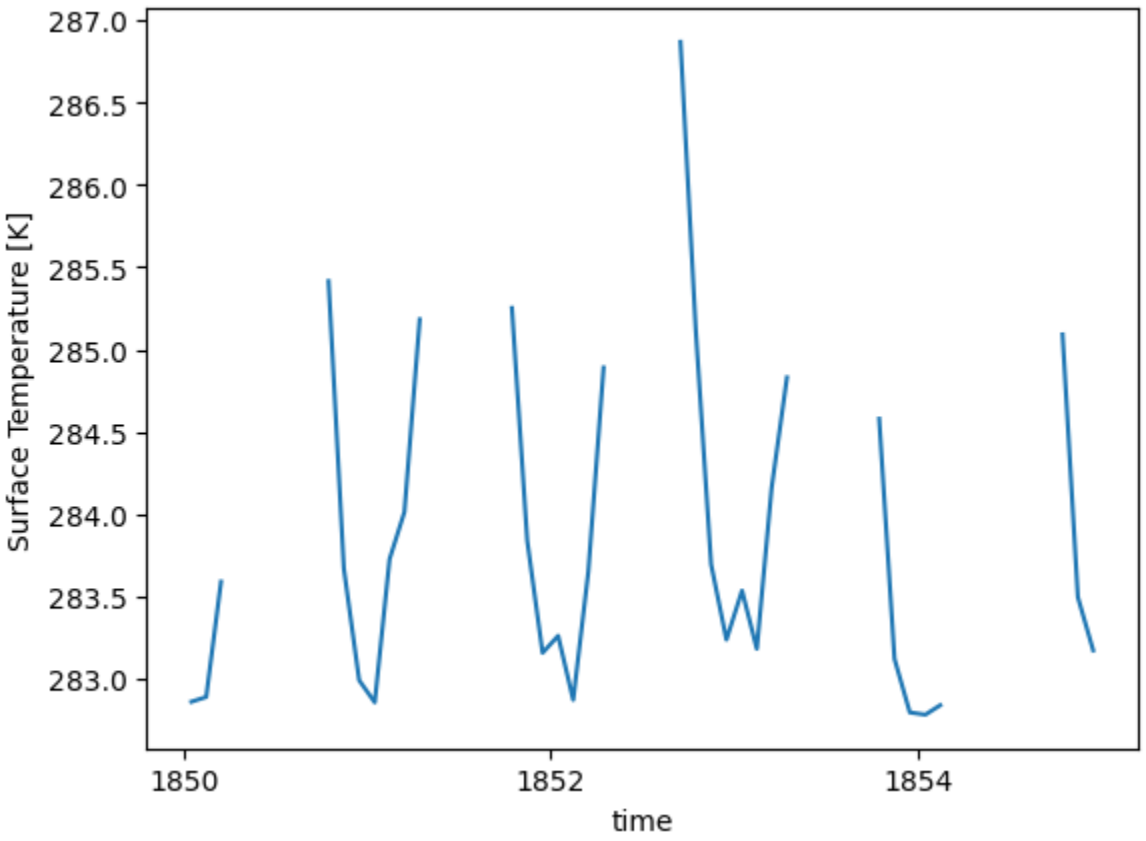
```
def calculate_spatial_average(
    ds: xr.Dataset,
    data_var: str,
    nan_threshold_fraction: float = 0.2,
    lat_dim: str = "lat",
    lon_dim: str = "lon",
    ) -> xr.DataArray:

    # Total number of elements along lat and lon dimensions for each time st
    total_elements = ds[data_var].sizes[lat_dim] * ds[data_var].sizes[lon_di

    # Count NaNs over the 'lat' and 'lon' dimensions for each time step
    nan_count = ds[data_var].isnull().sum(dim=[lat_dim, lon_dim])

    # Determine if the number of NaNs is less than certain fraction of the t
    nan_threshold_count = nan_threshold_fraction * total_elements
    valid_data_mask = nan_count < nan_threshold_count

    # Calculate the mean along the 'lat' and 'lon' dimensions, ignoring NaNs
    #spatial_mean = ds.spatial.average(data_var, skipna=True)[data_var]
    spatial_mean = ds.spatial.average(data_var)[data_var]

    # Set the average to NaN if the number of NaNs exceeds the threshold
    spatial_mean = spatial_mean.where(valid_data_mask, other=np.nan)
```

```
    return spatial_mean
```

In [ ]:
```
spatial_mean = calculate_spatial_average(ds, data_var, nan_threshold_fractio
```

In [ ]:
```
spatial_mean.plot()
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x155acaed0>]
```



Temporal average

In [ ]:
```
def calculate_temporal_average(
    ds: xr.Dataset,
    data_var: str,
    nan_threshold_fraction: float = 0.2,
    time_dim: str = "time",
    ) -> xr.DataArray:

    # Total number of elements along the time dimension
    total_elements = ds[data_var].sizes[time_dim]

    # Count NaNs over the 'time' dimension for each (lat, lon) point
    nan_count = ds[data_var].isnull().sum(dim=time_dim)

    # Determine if the number of NaNs is less than certain fraction of the t
    nan_threshold_count = nan_threshold_fraction * total_elements
    valid_data_mask = nan_count < nan_threshold_count

    # Calculate the mean along the 'time' dimension, ignoring NaNs
    #temporal_mean = ds.temporal.average(data_var, skipna=True)[data_var]
```

```
    temporal_mean = ds.temporal.average(data_var)[data_var]

    # Set the average to NaN if the number of NaNs exceeds the threshold
    temporal_mean = temporal_mean.where(valid_data_mask, other=np.nan)

    return temporal_mean
```

In [ ]:
```
temporal_mean = calculate_temporal_average(ds, data_var, nan_threshold_fract
```

In [ ]:
```
temporal_mean
```

Out[ ]:

xarray.DataArray 'ts' (**lat**: 145, **lon**: 192)

array([[nan, nan, nan, ..., nan,
nan, nan],
[nan, nan, nan, ..., nan,
nan, nan],
[nan, nan, nan, ..., nan,
nan, nan],
...,
[253.80782831, 253.83631185, 253.86972969, ..., 253.71189856,
253.74849125, 253.77323189],
[253.71469954, 253.72605014, 253.73871837, ..., 253.6769375 ,
253.6882096 , 253.70236836],
[253.6311178 , 253.6311178 , 253.6311178 , ..., 253.6311178 ,
253.6311178 , 253.6311178]])

▼ Coordinates:

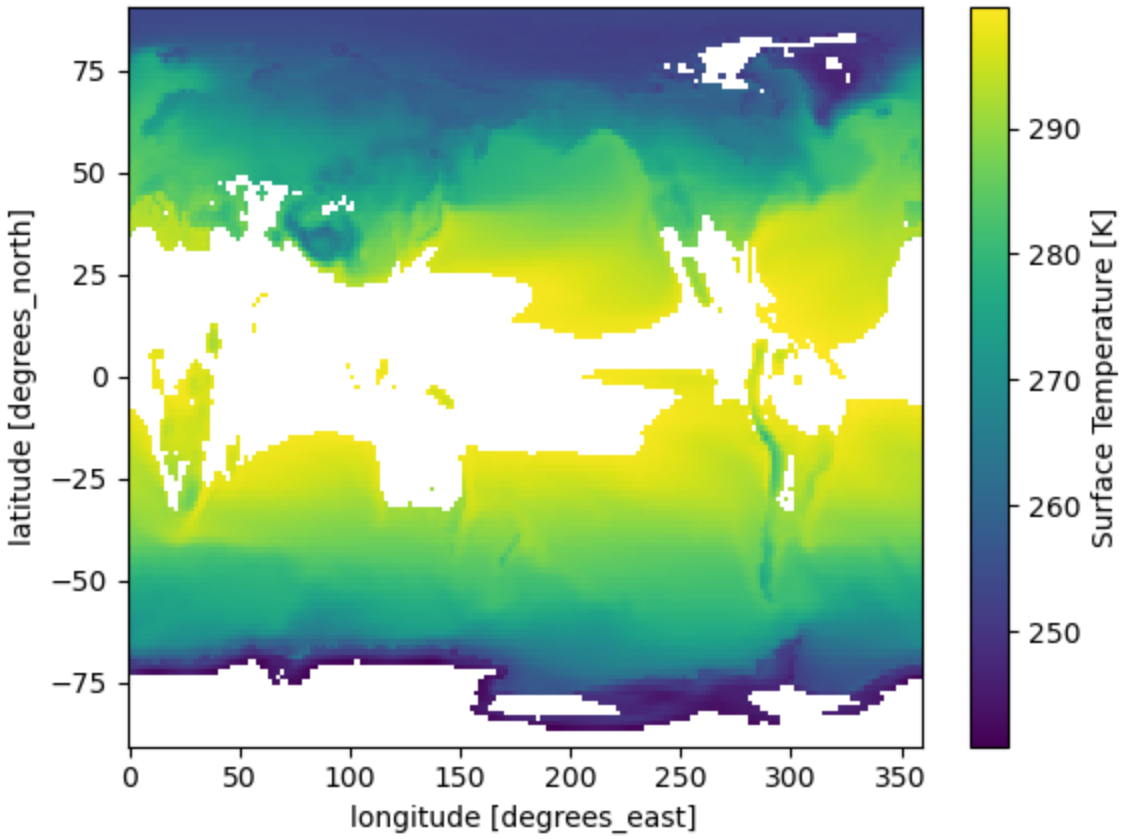
lat	(lat)	float64	-90.0 -88.75 -87.5 ... 88.75 90.0	
lon	(lon)	float64	0.0 1.875 3.75 ... 356.2 358.1	

▶ Indexes: (2)

▶ Attributes: (12)

In [ ]:
```
temporal_mean.plot()
```

Out[ ]:
```
<matplotlib.collections.QuadMesh at 0x155987050>
```


75
50
25
0
−25
−50
−75
latitude [degrees_north]
0
50
100
150
200
250
300
350
longitude [degrees_east]
290
280
270
260
250
Surface Temperature [K]

In [ ]: