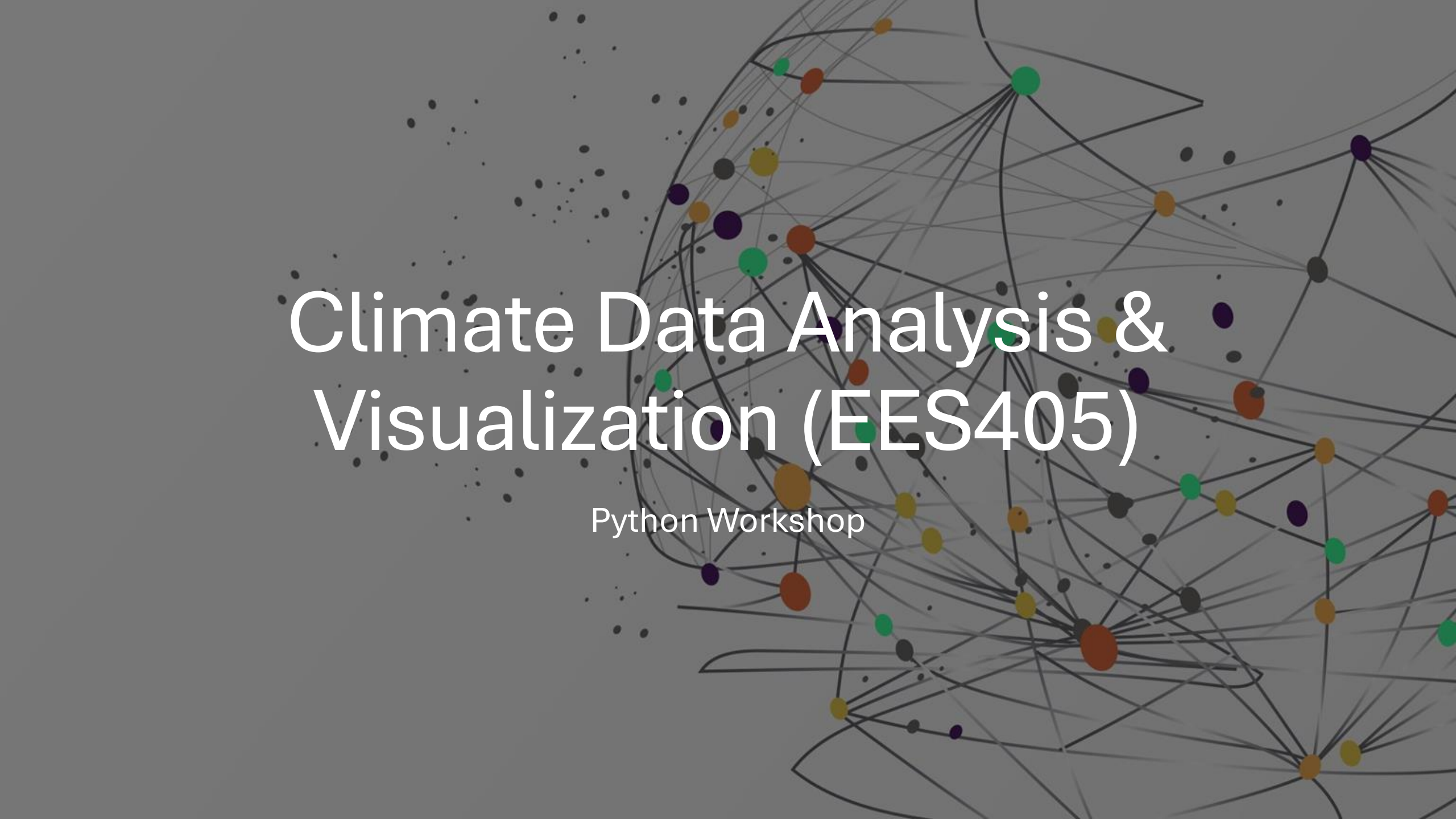


Climate Data Analysis & Visualization (EES405)

Python Workshop

Introduction to Python

- Python is a versatile programming language that's widely used in scientific computing and data analysis. Its simplicity and powerful libraries make it ideal for analyzing and visualizing climate data.

Importance of Climate Data Analysis

Climate data analysis helps us understand the past (climatology) , observe the present , and predict future climate patterns

Important in making informed decisions regarding the future.

Python Libraries for Climate Data

For climate-specific data analysis, we will use several libraries. Some of them are listed below (Non-exhaustive list)

- Xarray
- Numpy
- Matplotlib
- Cartopy
- Pandas
- Scipy

Can be installed via ‘conda ’ or ‘pip’

Xarray

Designed for working with labelled multi-dimensional arrays, making it perfect for climate data.

Facilitates advanced operations like group-by, resampling, and data alignment with minimal effort.

Integrates well with Dask for out-of-core computation on large datasets.

Provides tools to work seamlessly with data from netCDF files, a common format in climate science.

Numpy

Offers powerful N-dimensional array objects and a wide range of mathematical functions to operate on these arrays.

Serves as the foundational library for scientific computing in Python, underpinning many other libraries.

Efficient handling of large data sets with capabilities for linear algebra, Fourier transform, and random number capabilities.

Critical for performance-intensive computations needed in climate data analysis.

Matplotlib

A versatile plotting library capable of creating static, interactive, and animated visualizations in Python.

Provides a vast array of plot types, including line graphs, scatter plots, bar charts, and 3D visualizations.

Customizable and extendable, allowing for detailed adjustments to plot appearance and layout.

Essential for visualizing climate data trends, distributions, and relationships.

Cartopy

Geospatial data processing library for Python that enables map creation, manipulation, and visualization.

Supports a variety of map projections and geospatial operations to accurately represent the Earth's surface.

Facilitates the plotting of geospatial data for climate analysis, such as temperature patterns, storm tracks, and more.

Integrates with Matplotlib for creating publication-quality maps and figures.

Pandas

Provides high-performance, easy-to-use data structures, and data analysis tools.

Ideal for handling and analyzing time series data, a common format in climate datasets.

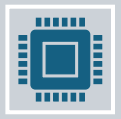
Offers data manipulation capabilities such as merging, reshaping, selecting, as well as data cleaning.

Enables quick and efficient data import/export from various file formats

Scipy

A library for scientific and technical computing with modules for optimization, linear algebra, integration, interpolation, and more.

Offers tools for statistical analysis and modeling, which are valuable for hypothesis testing and data exploration in climate studies.



Includes specialized modules for signal processing and image manipulation, useful for analyzing spatial and temporal changes in climate data.

Works closely with NumPy arrays and provides efficient numerical routines such as numerical integration and optimization.

Practical Examples

We'll explore practical examples of climate data analysis, including temperature trend analysis, visualization of climate patterns, and working with datasets.

Let’s Begin !

Step 1: Import the Libraries

We begin by loading the required libraries in our environment

```
import numpy as np
import matplotlib.pyplot as plt
import xarray as xr
import cartopy as cp
import cartopy.crs as ccrs
import cartopy.feature as cfeature
```

Step 2: Load the data

We load the dataset into our environment to start the analysis

```
file = '/media/path/NCEP/2d/air.2m.mon.mean.nc’
data = xr.open_dataset(file)
```

Step 3: Exploratory Analysis (Optional)

We can try and do some exploratory analysis on the dataset to learn more about the type of data you are handling

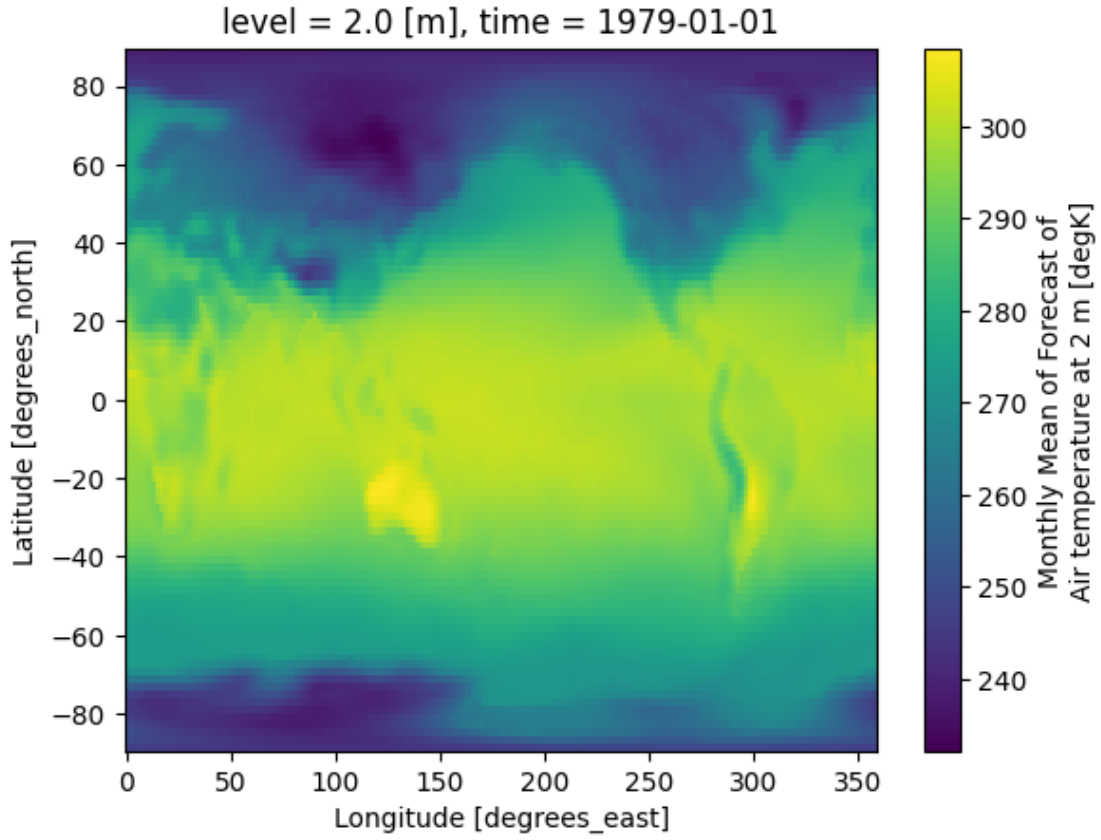
```
print(data.air.attrs) # print out the attributes of the air variable
print(data.air.dims) # print out the dimensions of the air variable
print(data.air.coords) # print out the coordinates of the air variable
print(data.air.attrs['units']) # print out the units of the air variable
```

```
{'long_name': 'Monthly Mean of Forecast of Air temperature at 2 m', 'units': 'degK', 'precision': 2, 'GRIB_id': 11, 'GRIB_n
ame': 'TMP', 'var_desc': 'Air temperature', 'dataset': 'NCEP/DOE AMIP-II Reanalysis (Reanalysis-2) Monthly Averages', 'leve
l_desc': '2 m', 'statistic': 'Mean', 'parent_stat': 'Individual Obs', 'standard_name': 'air_temperature', 'cell_methods':
'time: mean (interval: 6 hours to daily) time: mean (interval: 1 day to monthly)', 'valid_range': array([120., 430.], dtype
=float32), 'actual_range': array([197.63, 313.76], dtype=float32)}
('time', 'level', 'lat', 'lon')
Coordinates:
  * level    (level) float32 2.0
  * lat      (lat) float32 88.54 86.65 84.75 82.85 ... -84.75 -86.65 -88.54
  * lon      (lon) float32 0.0 1.875 3.75 5.625 7.5 ... 352.5 354.4 356.2 358.1
  * time     (time) datetime64[ns] 1979-01-01 1979-02-01 ... 2023-01-01
(529, 1, 94, 192)
degK
```

Step 4a: Plot the spatial data

Let’s try and plot the spatial plot of the data we have loaded

```
# spatial data
data.air.isel(time=0).plot() #plot
```

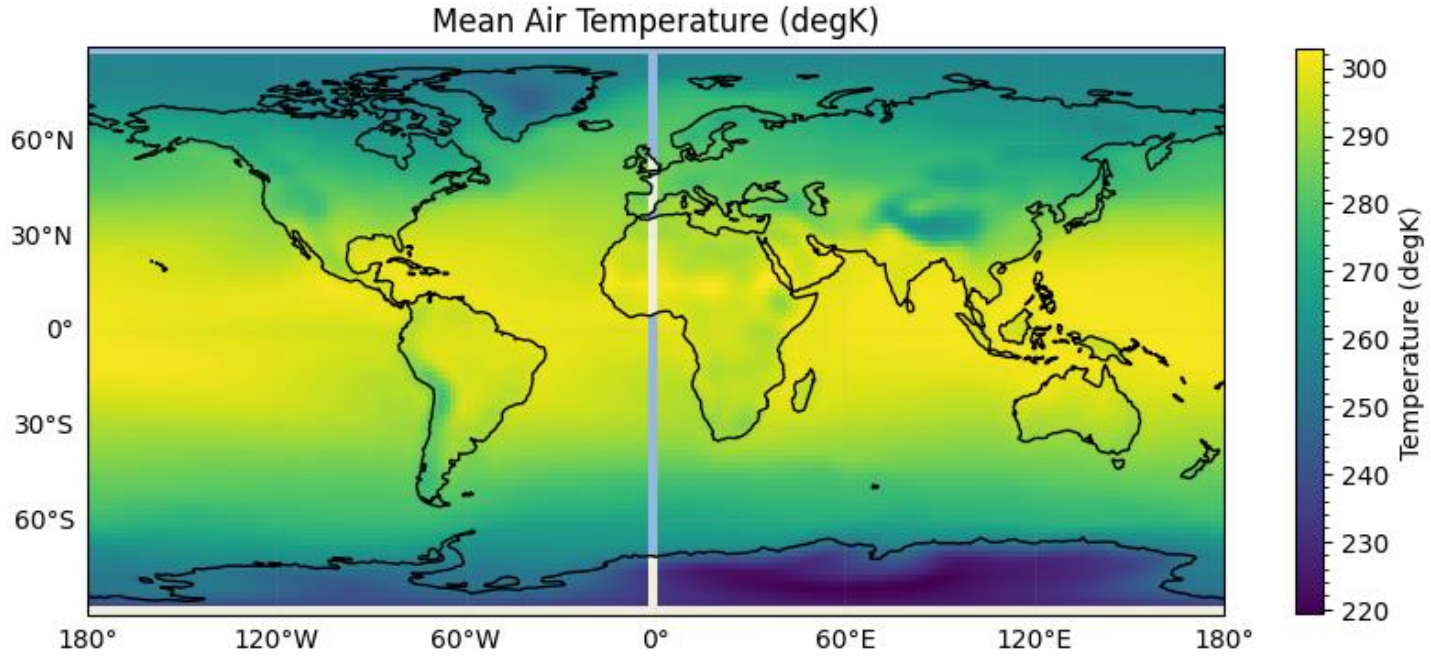


Not pretty right !

Step 4b: Plot the spatial data (pretty one !)

Let's try and plot the spatial plot of the data (the pretty one!)

```
# set the projection and plot
ax = fig.add_subplot(1,1,1,projection=ccrs.PlateCarree(central_longitude=0.0, globe=None)) # set the projection
mp = ax.imshow(data_mean,extent=(lon.min(),lon.max(),lat.min(),lat.max()),cmap='viridis')
```

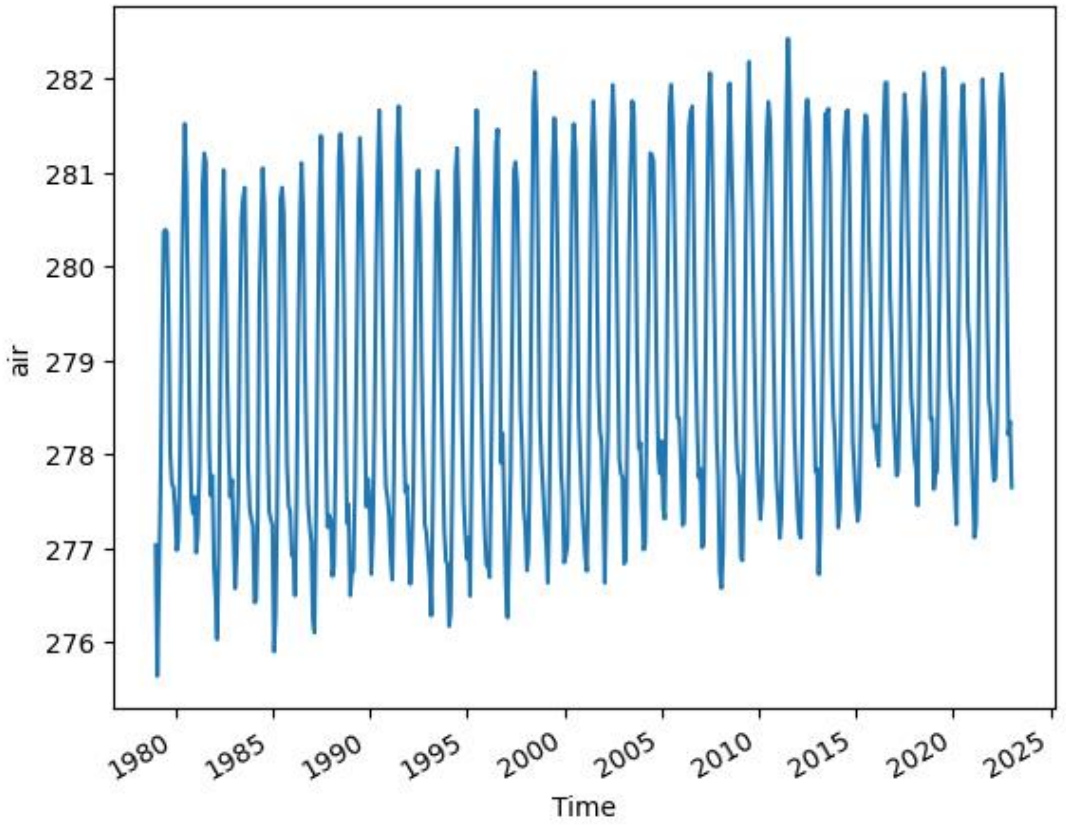


Step 5a: Plot the temporal data

Let’s try and plot the timeseries of the data we have loaded.

```
air_fldmean = data.air.mean(dim=['lat','lon','level'])
air_fldmean.plot()
```

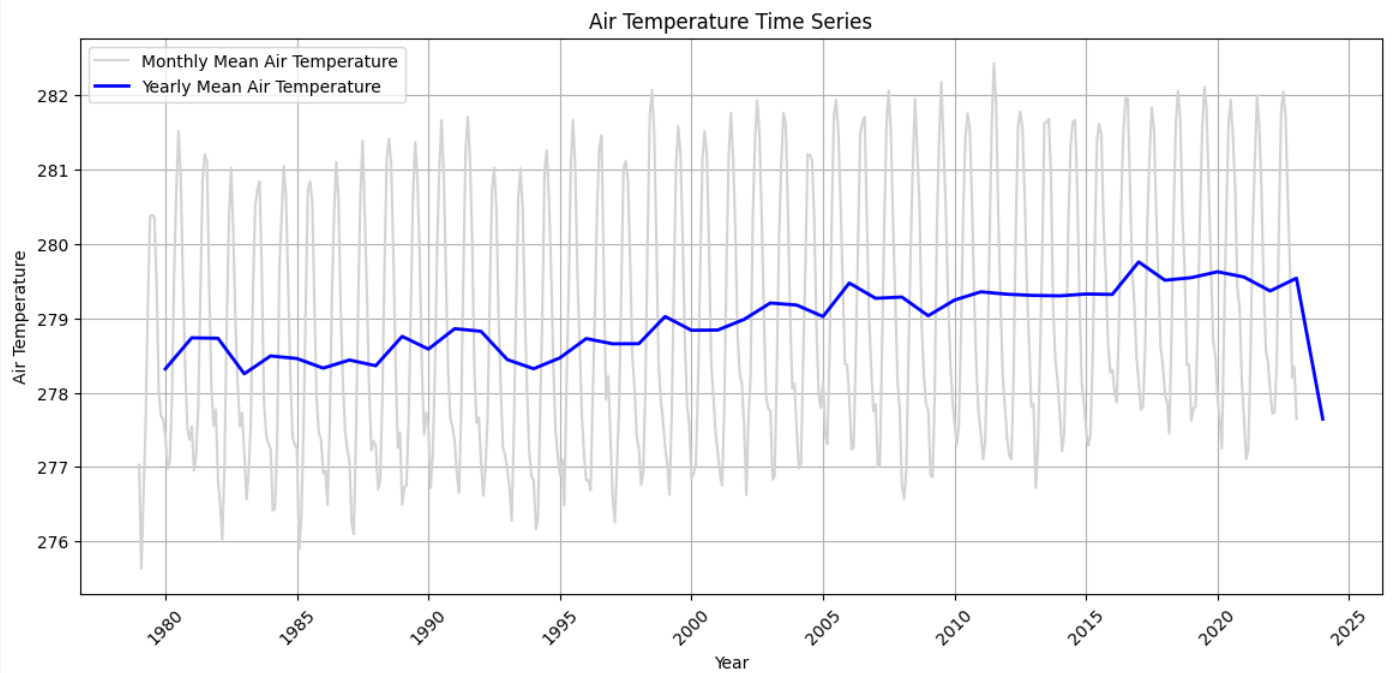
A bit cluttered right !



Step 5b: Plot the temporal data (Clean plot !)

Let’s try and plot the timeseries of the data in a much cleaner and more understandable way

```
# lets resample the data to yearly means
air_fldmean_yearly = air_fldmean.resample(time='Y').mean()
```

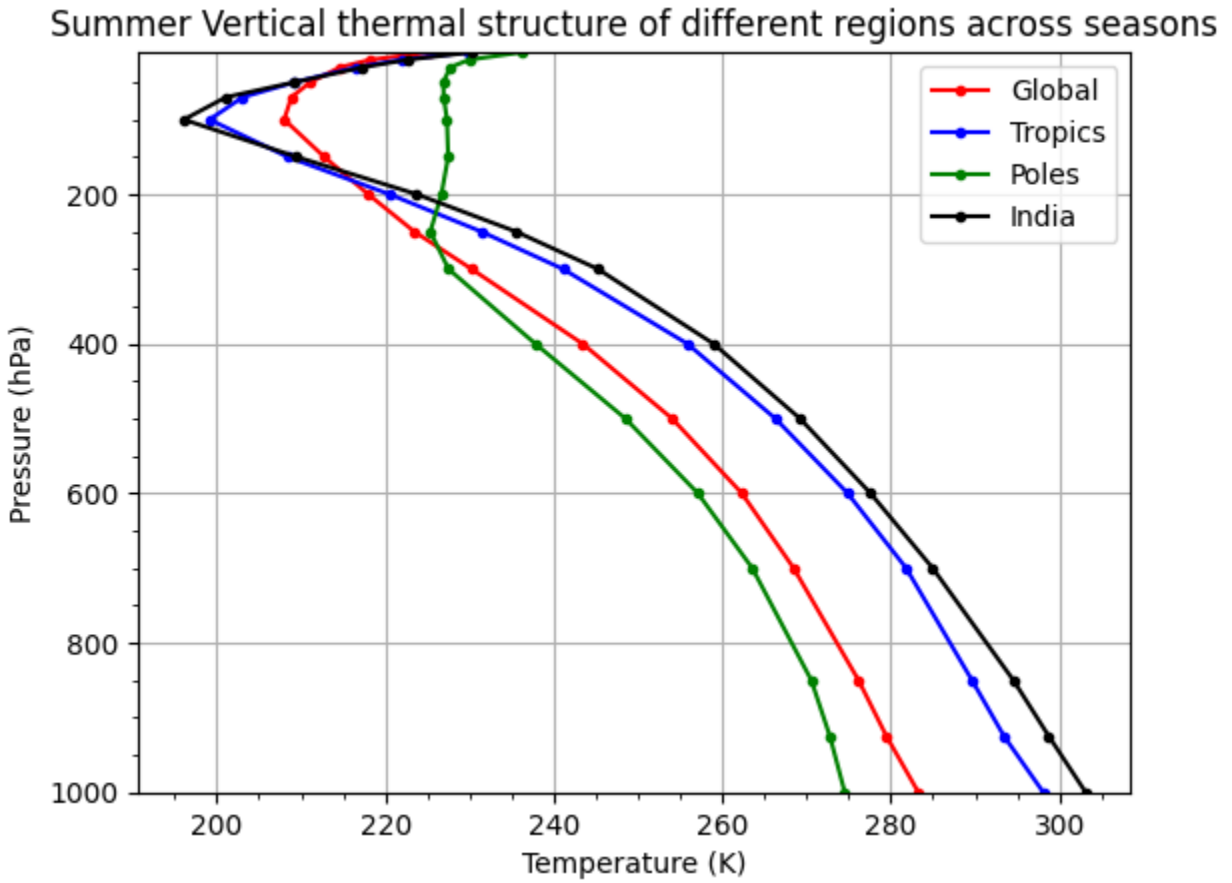


Step 6: Plot the levels data (Vertical Series)

Let’s try and plot the temperature along the pressure levels

```
data_levels =
data_3d.air.mean(dim=['lat','lon','time'])
# reduce the dimensionality of the data
```

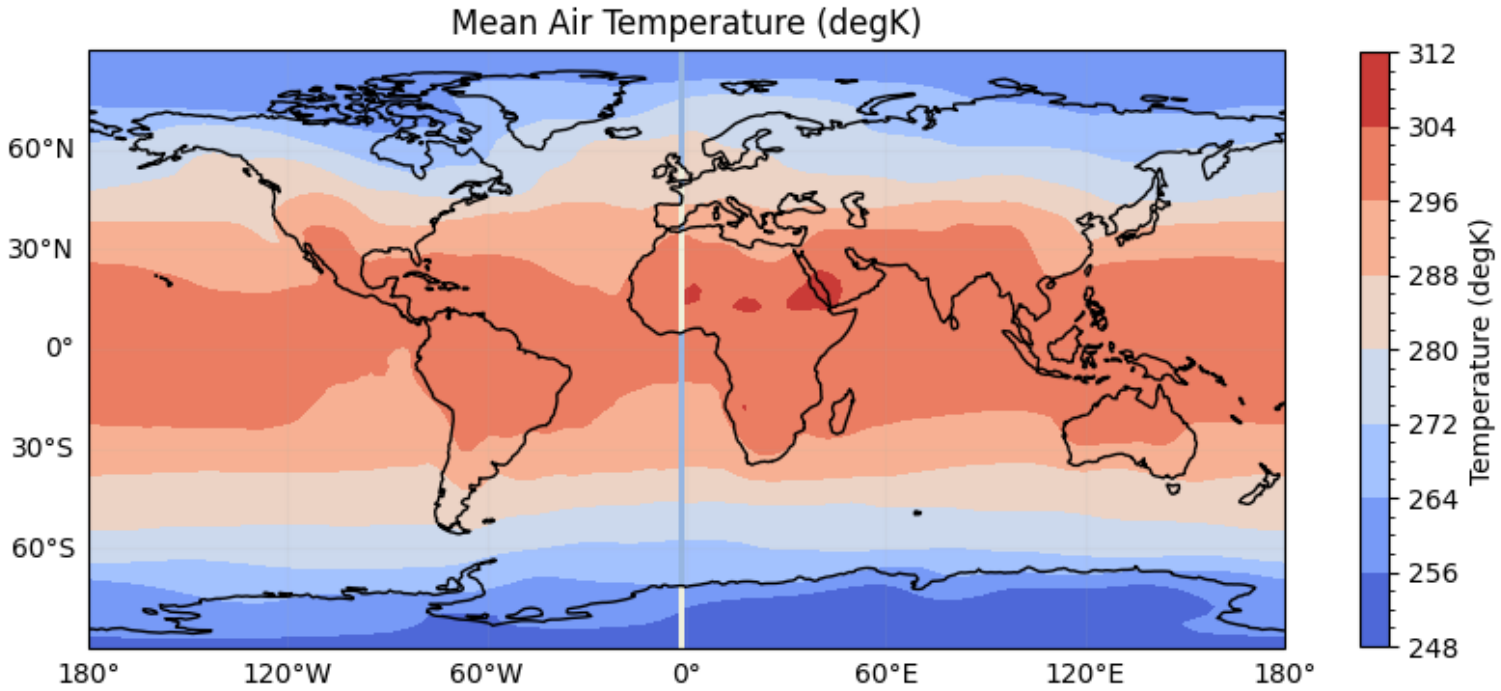
Something familiar looking !



Step 7: Plot the contours

Let’s try and plot the contours plot of a climate variable

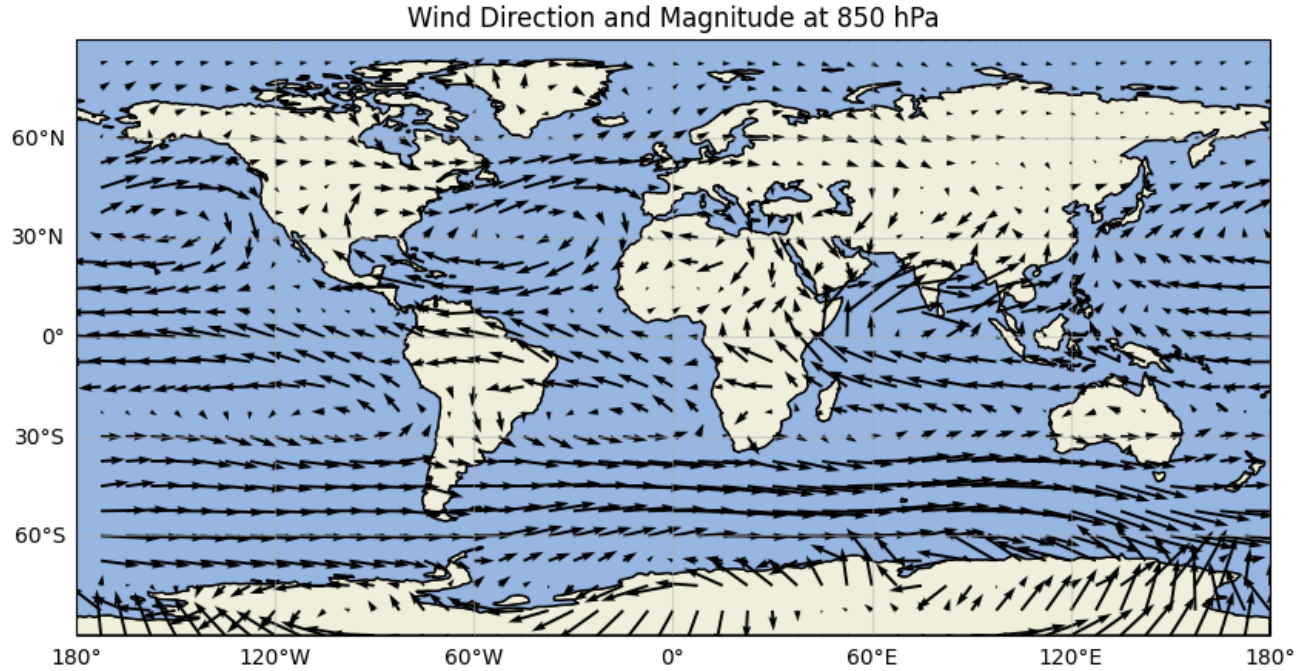
```
mp = ax.contourf(lon, lat,
data_contour_mean.sel(level=1000),transform=ccrs.PlateCarree(),cmap='
coolwarm')
```



Step 8a: Plot the quiver plot

Let’s try to do the quiver plot (let’s say wind data)

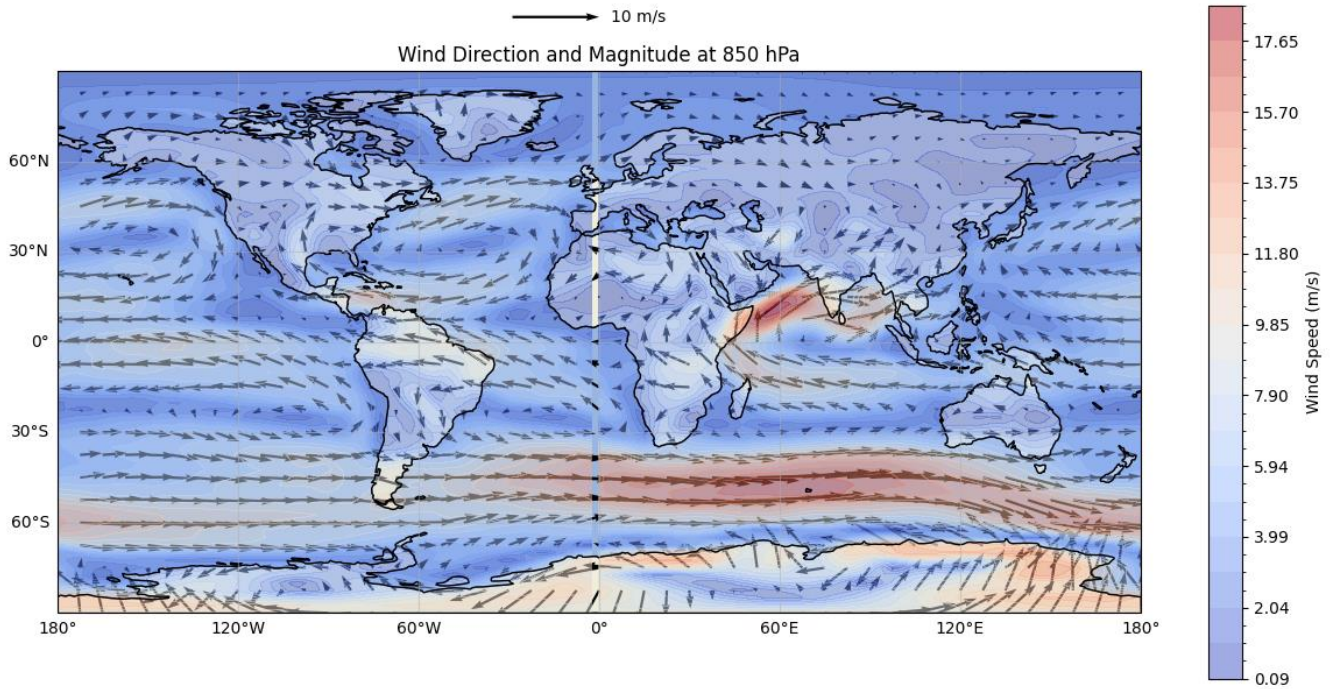
```
quiver_skip = 3
qv = ax.quiver(uwnd_850.lon[::quiver_skip], uwnd_850.lat[::quiver_skip],
               uwnd_850[::quiver_skip, ::quiver_skip], vwnd_850[::quiver_skip,
::quiver_skip],transform=ccrs.PlateCarree(), scale=250)
```



Step 8b: Plot the quiver plot with contours

Let’s try and add some contours background

```
levels = np.linspace(wind_speed.min(), wind_speed.max(), 20)
cp = ax.contourf(uwnd_850.lon, uwnd_850.lat, wind_speed, levels=levels,
                 transform=ccrs.PlateCarree(), cmap='coolwarm', alpha=0.5)
```



Thank You !