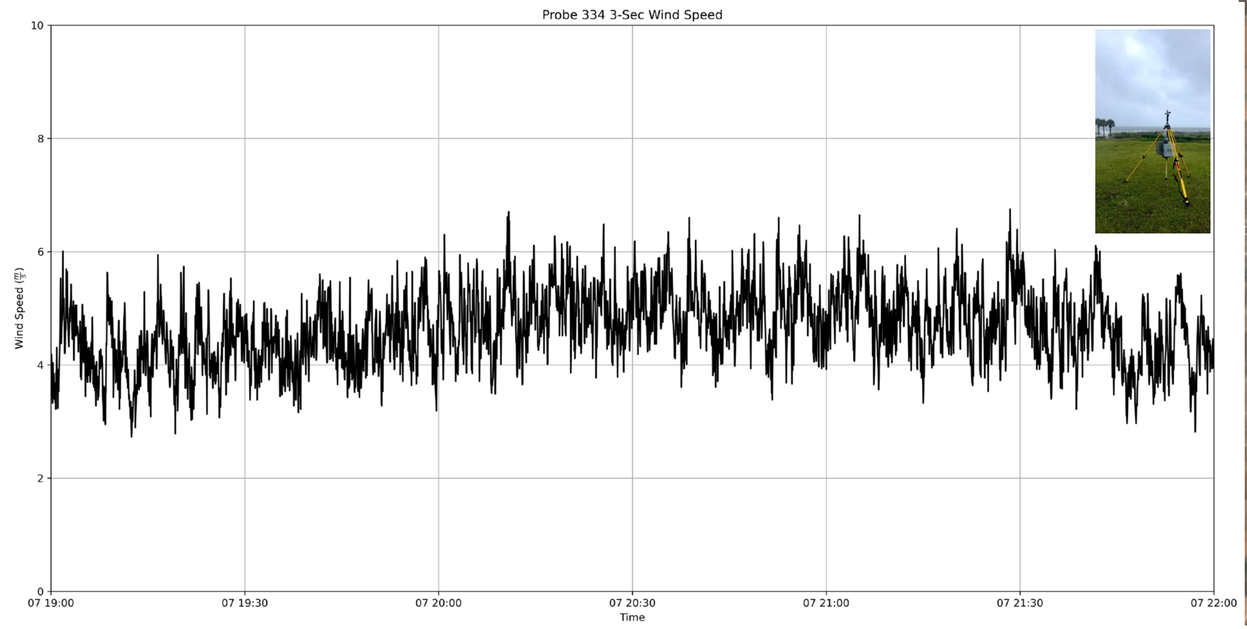
Large Dataset Manipulation in Python: A Multi-Sensor, Multi-Model Case Study

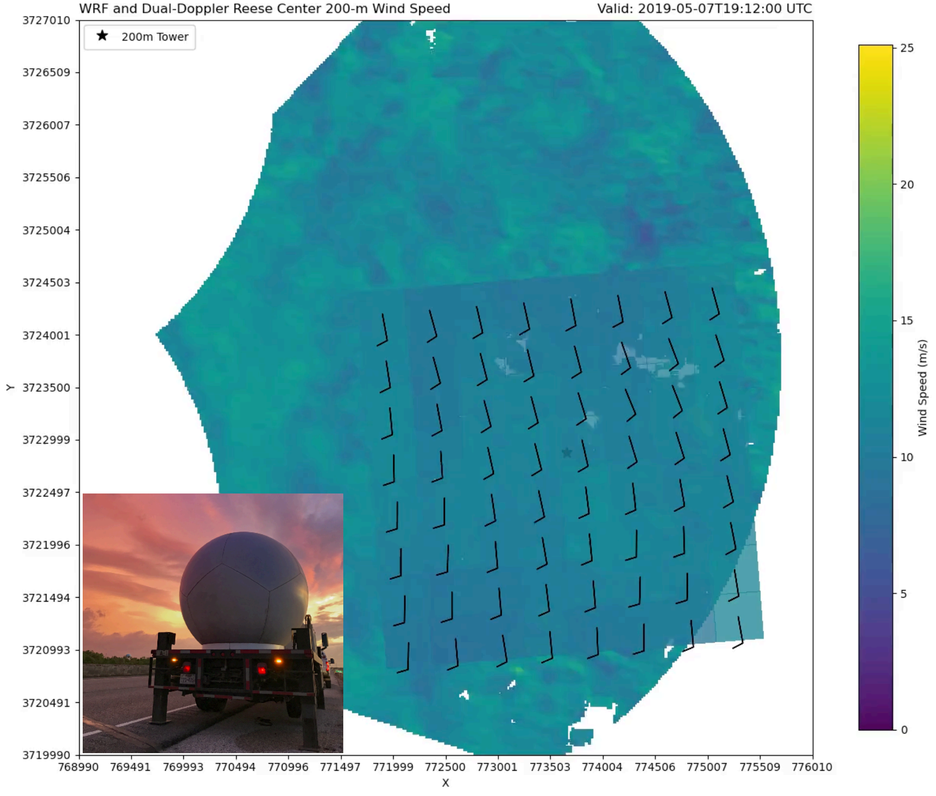
Tyler Wixtrom

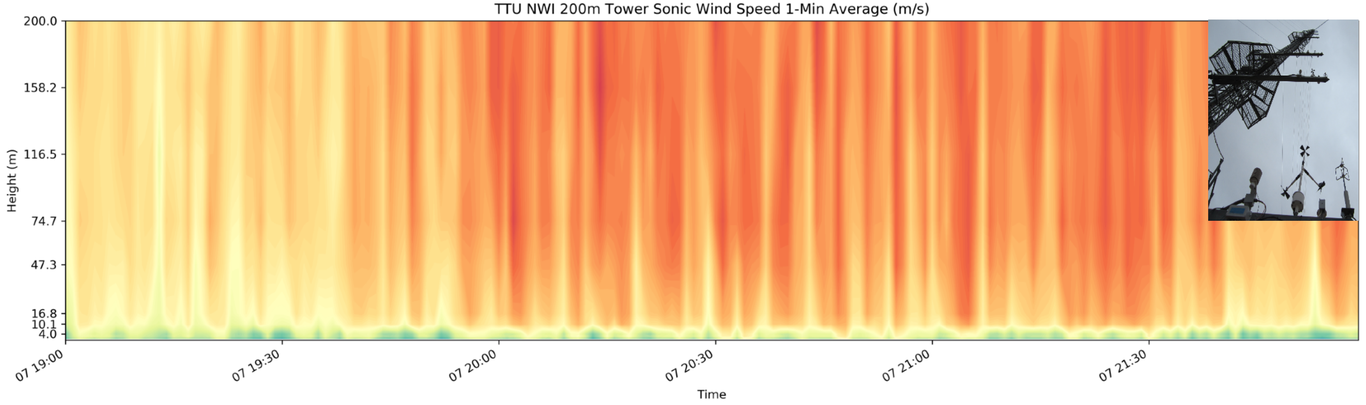
Background

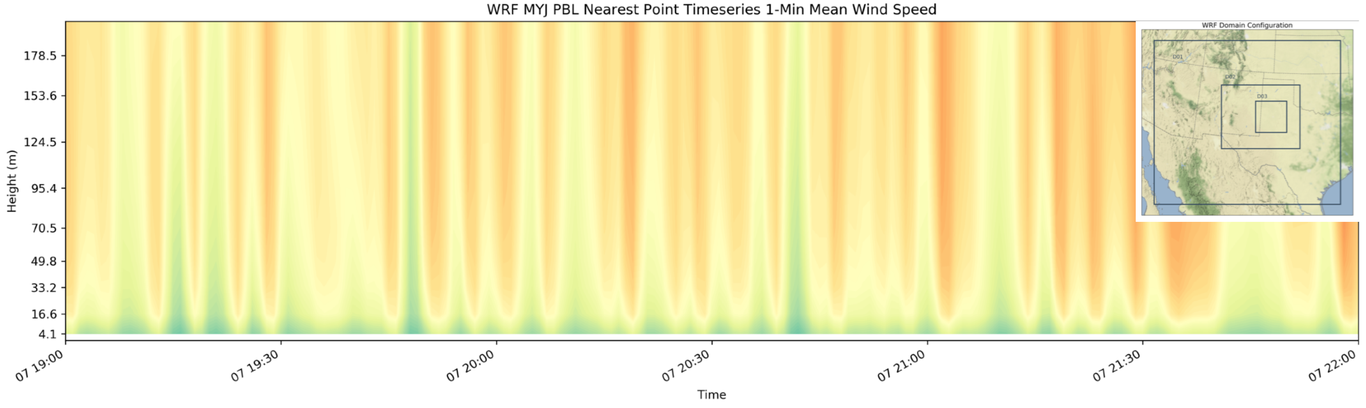
- Analysis of large datasets is common within the earth sciences
  - Multi-dimensional
  - Multivariate
  - Multi-model
  - Multi-sensor
- Total dataset sizes often exceed several TB


Probe 334 3-Sec Wind Speed
Wind Speed (m/s)
Time
07 19:00
07 19:30
07 20:00
07 20:30
07 21:00
07 21:30
07 22:00


KA-1
KA-2
0328C
0329C
0330C
0325C
0326C
0441C
0438C
0334C
0446C
0445C
0333C
0447C
0327C
0331C
Reese Center
0444C
0332C
Shallowater


WRF and Dual-Doppler Reese Center 200-m Wind Speed
Valid: 2019-05-07T19:12:00 UTC
200m Tower
Wind Speed (m/s)
X
Y


TTU NWI 200m Tower Sonic Wind Speed 1-Min Average (m/s)
Height (m)
Time
Wind Speed (m/s)


WRF MYJ PBL Nearest Point Timeseries 1-Min Mean Wind Speed
WRF Domain Configuration
Height (m)
Time
Variance (m/s)

Challenges

- Data formats and metadata conventions
- Computation with large datasets
- Unit conversions
- Aligning datasets in space and time

Formats

- Datasets are often in a variety of formats
  - NetCDF3, NetCDF4, GRIB, Zarr, Text, etc.
- It is often best to convert all datasets to a common format prior to analysis

	A	B	C	D	E	F	G	H	I	J
1	3.35541	5.97263	0.671082	95.828	78.0188	53.198	26.403335	4.004123	4.585727	-1.901399
2	3.176455	5.905522	1.319795	95.72	78.0152	53.186	26.404162	4.921268	4.764682	-2.080354
3	2.572481	5.525242	1.655336	95.648	78.0152	53.216	26.403099	4.675205	5.681828	-2.147462
4	2.326418	5.927891	1.565858	95.648	78.098	53.218	26.402744	3.869906	6.062107	-1.11847
5	2.61722	6.017369	1.431642	95.612	78.0872	53.21	26.403807	2.885653	5.95026	-0.917145
6	2.23694	6.218693	1.87903	95.666	78.08	53.228	26.402981	3.802798	6.062107	-1.185578
7	1.789552	6.35291	2.214571	95.756	77.9972	53.212	26.404516	2.863283	6.241063	-2.080354
8	2.057985	5.883152	1.968507	95.774	78.0152	53.212	26.403807	3.802798	5.860783	-2.527742
9	2.527742	6.397648	1.319795	95.666	78.0044	53.176	26.40487	3.511996	5.748936	-2.505373
10	1.923768	6.062107	0.782929	95.6	77.9828	53.178	26.401799	4.250186	4.921268	-2.840914
11	1.051362	5.681828	-0.156586	95.504	78.0584	53.352	26.40239	3.981753	4.608096	-2.460634
12	1.051362	5.883152	-0.850037	95.468	78.1232	53.236	26.40617	3.467257	4.317294	-2.460634
13	0.626343	5.480503	-0.469757	95.414	78.0404	53.234	26.402508	2.326418	4.719943	-2.259309
14	0.603974	5.055484	-0.290802	95.396	78.0188	53.206	26.403689	2.9975	4.630466	-2.23694
15	0.35791	4.339664	0.089478	95.342	78.05	53.24	26.404398	3.310671	4.85416	-2.192201
16	-0.335541	4.026492	0.290802	95.414	78.1088	53.244	26.405461	3.444888	4.85416	-2.080354
17	-1.252686	3.467257	-0.0223							
18	-1.386903	3.534365	-0.5144							
19	-0.872407	3.422518	-0.760							
20	-1.543489	3.444888	-0.7158							
21	-0.648713	3.959384	-0.8947							
22	-0.134216	3.937014	-0.6934							
23	0.156586	4.071231	0.2908							
24	0.536866	3.71332	-0.0447							
25	0.73819	3.601473	-0.8052							
26	1.342164	3.981753	-0.8724							
27	1.051362	4.205447	-0.380							
28	0.73819	4.0936	-0.1565							
29	0.201325	3.646212	-0.5816							
30	0.313172	3.35541	-0.7158							

```
[2]: data
```

```
[2]: xarray.Dataset
```

▶ Dimensions: (**level**: 10, **time**: 540024)

▼ Coordinates:

time	(time)	datetime64[ns]	2019-05-07T19:00:00 ... 2019-05-...
level	(level)	float64	0.9 2.4 4.0 ... 116.5 158.2 200.0

▼ Data variables:

sonic_u_compo...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
sonic_v_compo...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
sonic_w_compo...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
sonic_temperat...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
temperature	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
relative_humidity	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
barometric_pre...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
sonic_along_wi...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
sonic_cross_wi...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
sonic_vertical_...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
sonic_wind_spe...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....
sonic_wind_dire...	(level, time)	float64	dask.array<chunksize=(10, 10000), meta=np....

▼ Attributes:

history :	Processed from .csv by Tyler Wixtrom
created :	National Wind Institute, Texas Tech University

Xarray

- Support for a variety of input formats
- Multiple variables linked as dataset
- Metadata and coordinate information stored with dataset
- Support for out of memory subsetting
- Robust support for NetCDF

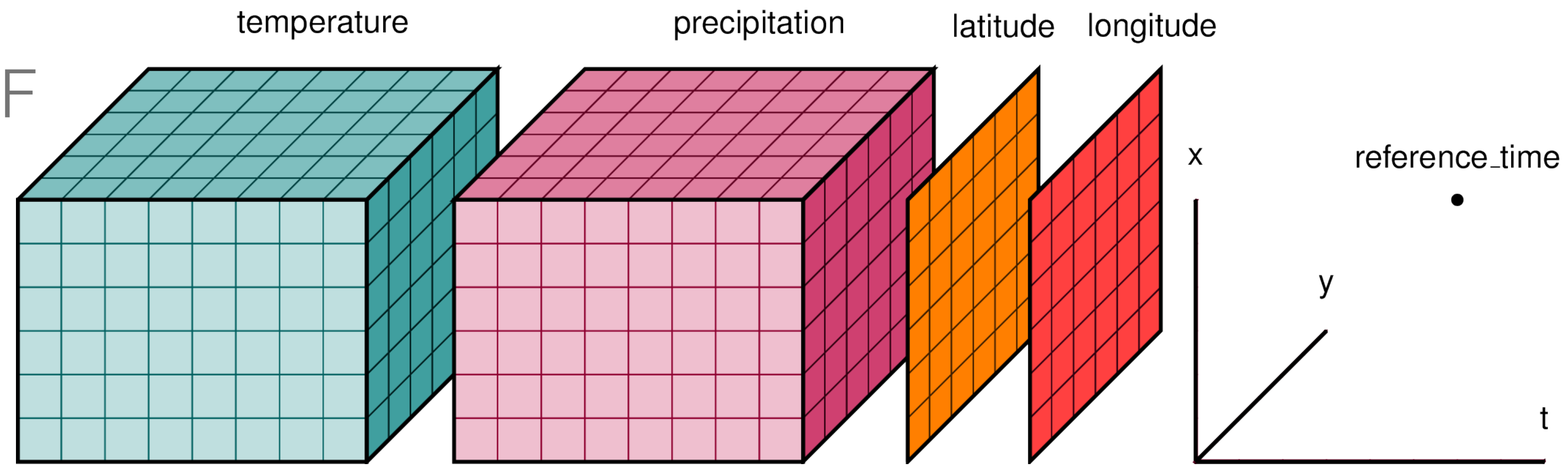
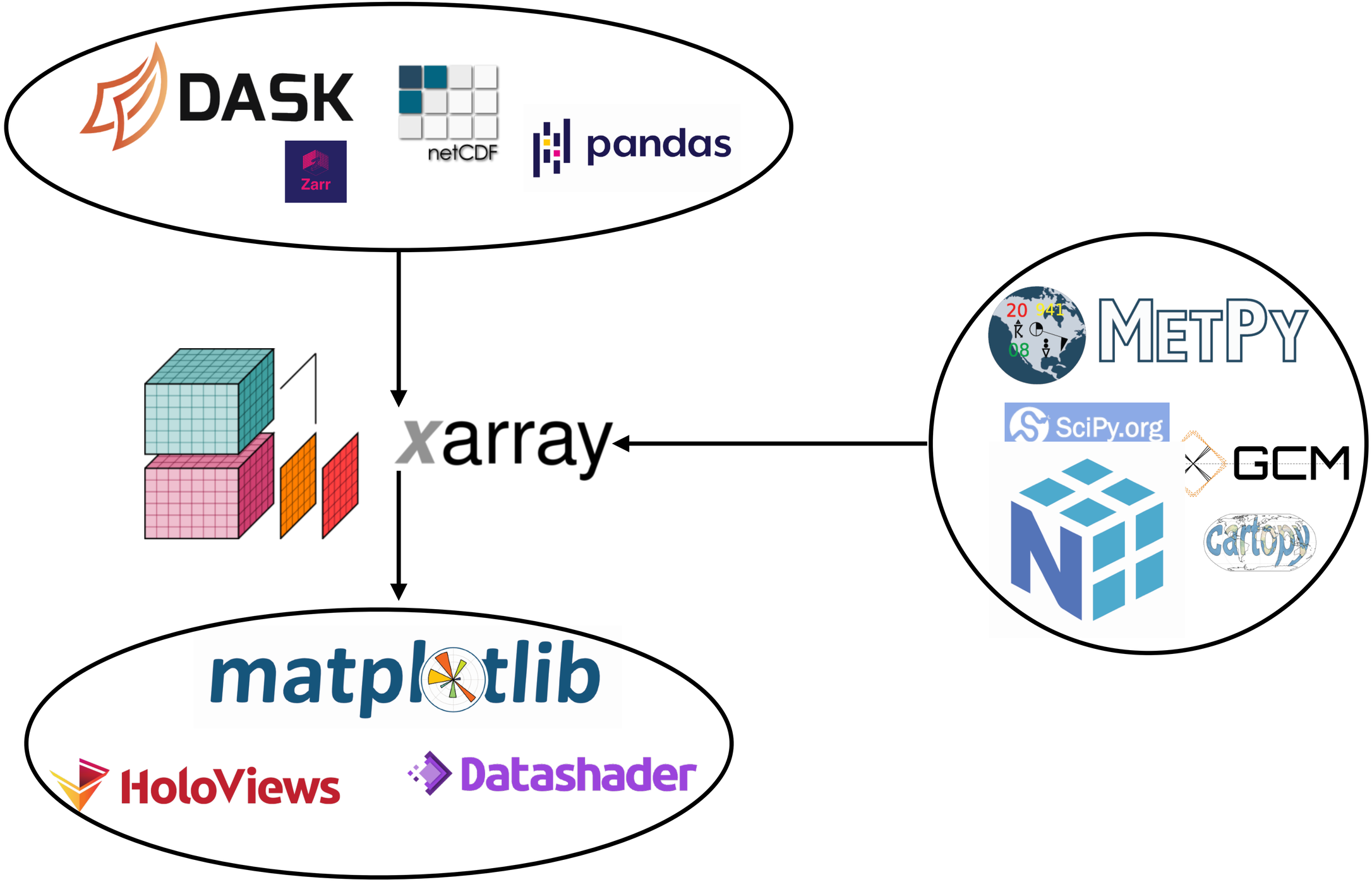


Image Credit: http://xarray.pydata.org/en/stable/data-structures.html


DASK
Zarr
netCDF
pandas
xarray
MetPy
SciPy.org
GCM
cartopy
matplotlib
HoloViews
Datashader

Computations

- Array computations are typically performed through vectorized operations to limit the use of loops
- For large arrays, this can easily overwhelm available memory
- Operations are serial and do not take advantage of additional processors

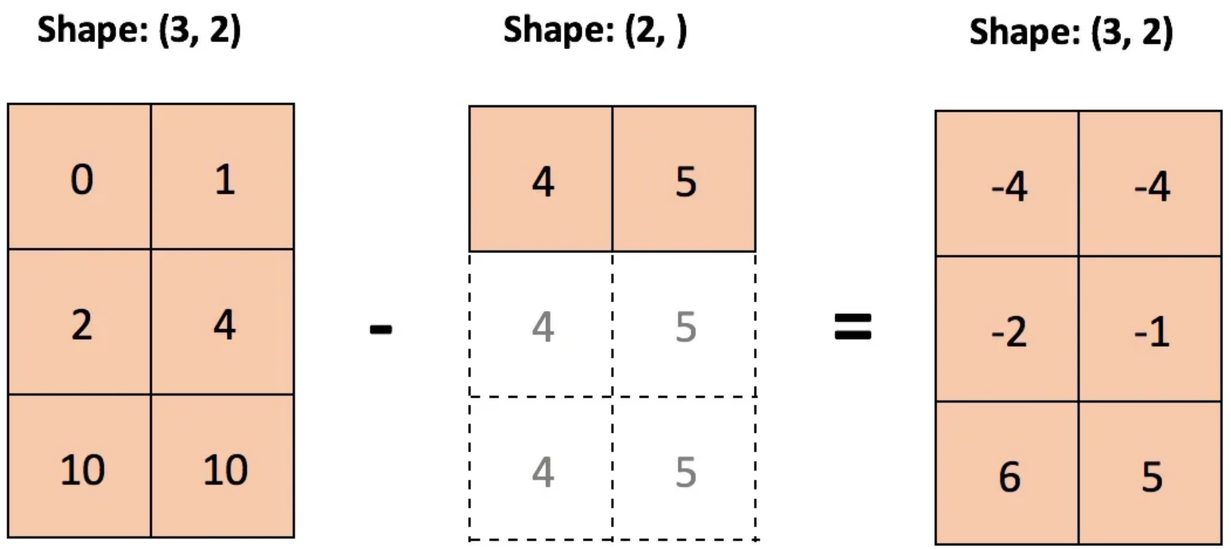


Image Credit: https://realpython.com/numpy-array-programming/

DASK

- Array library for high performance computing
- Based on familiar user interfaces (NumPy, Pandas)
- Support for lazy (out-of-memory) subsetting and parallel computations
- Support for a variety of HPC, cloud, and local configurations
- Integrates with existing analysis and visualization tools (sort of)

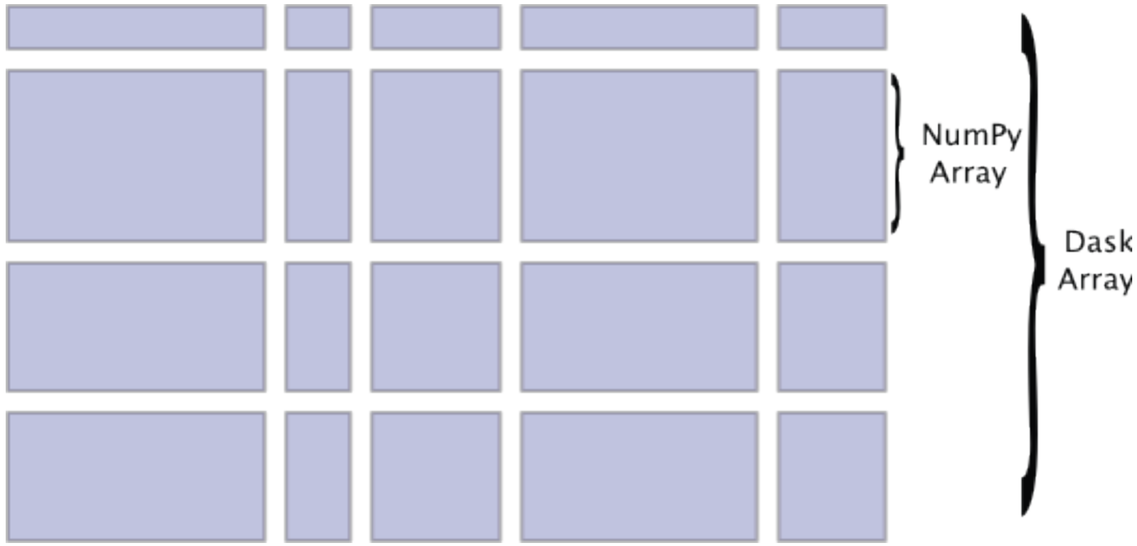


Image Creadit: https://docs.dask.org/en/latest/array.html

Unit Conversions

- Unit support is a longstanding problem
- Pint integrates with the NumPy array to create a quantity with units and conversion mapping
  - Implemented in MetPy as **`metpy.units`**

```
from metpy.units import units
temperature = (temp * units('degF')).to('kelvin')
```

- Robust support for NumPy-based workflows
- Xarray and Dask are not currently supported (yet)

Space and Time Conversions

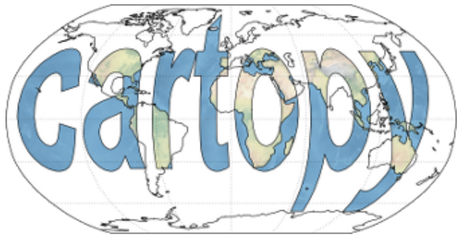
- Datasets should be aligned in space and time for analysis
- For unprojected data, this can be easily done within Xarray

```
import xarray as xr
data = xr.open_dataset('datafile.nc')
ws = data['wind_speed']
ws_1min = ws.resample(time='1Min').mean(dim='time')
ws_10m = ws.interp(height=10)
```

Space and Time Conversions

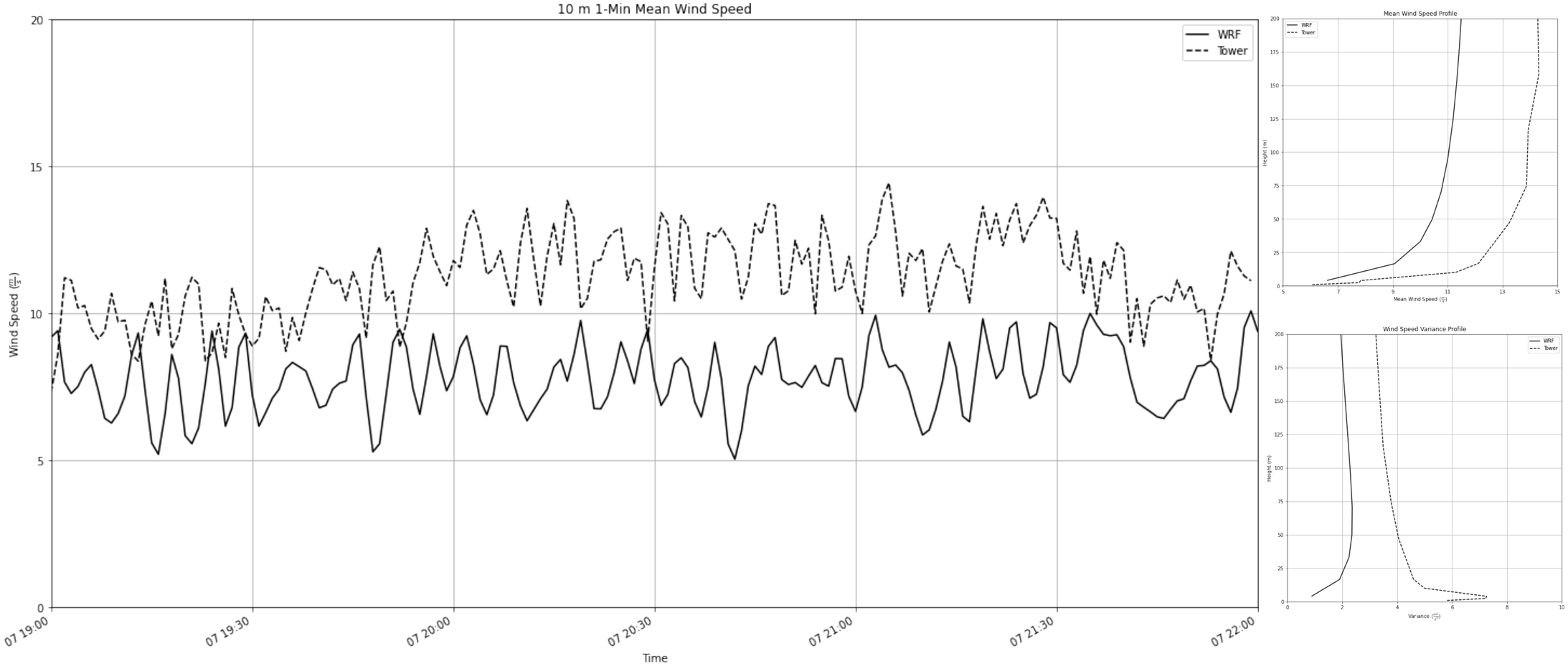
- For projected data, use CartoPy
  - Tip: Store projection information within Xarray for easy recall

```
import xarray as xr
import matplotlib.pyplot as plt
import cartopy.crs as ccrs

# open data
wrf = xr.open_dataset('model.nc')

# define data projection
globe = ccrs.Globe(ellipse=wrf.ellipse, semimajor_axis=wrf.semimajor_axis, semiminor_axis=wrf.semiminor_axis)
wrfcrs = ccrs.LambertConformal(central_longitude=wrf.standard_longitude,
                               central_latitude=wrf.central_latitude,
                               standard_parallels=(wrf.true_latitude_1, wrf.true_latitude_2),
                               globe=globe)

# define plot projection
plotcrs = ccrs.UTM(13)

# make the figure
fig = plt.figure(figsize=(15,15))
ax = fig.add_subplot(projection=plotcrs)
ax.pcolormesh(wrf.x, wrf.y, wrf.wind_speed, transform=wrfcrs)
plt.show()
```



Code Example

Slides and code example available at https://github.com/tjwixtrom/AMS-Python-2021



Contact

tyler.wixtrom@ttu.edu

tjwixtrom

@TWixtrom

Image Credit: Anna Thomas, TTU National Wind Institute