https://unidata.github.io/python-training/workshop/MetPy_Case_Study/metpy-case-study/

Retrieve data from National Center for Environmental Information (NCEI, formerly NCDC) then plot with Python.

This is a tutorial on building a case study map for Dynamic Meteorology courses with use of Unidata tools, specifically MetPy and Siphon. In this tutorial we will cover accessing, calculating, and plotting model output.

Let's investigate The 2016 Blizzard, although it would easy to change which case you wanted (please feel free to do so).

Reanalysis Output: NARR 00 UTC 23 Jan 1996

Data from Reanalysis on pressure surfaces:

- 2m Temperature

```
from datetime import datetime

import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
import metpy.calc as mpcalc
import numpy as np

from metpy.plots import StationPlot
from metpy.units import units
from netCDF4 import Dataset, num2date
from scipy.ndimage import gaussian_filter
from siphon.catalog import TDSCatalog
```

Case Study Data

There are a number of different sites that you can utilize to access past model output analyses and even forecasts. The most robust collection is housed at the National Center for Environmental Information (NCEI, formerly NCDC) on a THREDDS server. The general website to begin your search is

https://www.ncdc.noaa.gov/data-access

this link contains links to many different data sources (some of which we will come back to later in this tutorial). But for now, lets investigate what model output is available

https://www.ncei.noaa.gov/products/weather-climate-models

Retrieve NARR data from National Centers for Environmental Information

North American Regional Reanalysis (NARR) NARR is a regional reanalysis of North America containing temperatures, winds, moisture, soil data, and dozens of other parameters at 32km horizontal resolution.

Lets investigate what specific NARR output is available to work with from National Centers for Environmental Information (NCEI).

https://www.ncei.noaa.gov/products/weather-climate-models/north-american-regional

We specifically want to look for data that has "TDS" data access, since that is short for a *THREDDS* server data access point. Let's go ahead and use the NARR Analysis data to investigate the past case (The 1996 Blizzard) we identified:

https://www.ncei.noaa.gov/thredds/catalog/model-narr-a-files/199601/19960123/catalog.html?dataset=model-narr-a-files/199601/19960123/narr-a_221_19960123_0000_000.grb

And we will use a Unidata python package called Siphon to read this data through the THREDDS data server NetCDF Subset Service link:

https://www.ncei.noaa.gov/thredds/ncss/model-narr-a-files/199601/19960123/narr-a_221_19960123_0000_000.grb/dataset.html

```
# Case Study Date
year = 1996
month = 1
day = 24
hour = 0

dt = datetime(year, month, day, hour)
```

The following code chunk automate the spatial, temporal, and variable subsetting operation on the THREDDS data server for the 00 UTC, Jan 23, 2016 NARR data. The data is output into the Network Common Data Form (NetCDF) format, a standard for sharing scientific data in the weather and climate community.

```
# Read NARR Data from THREDDS server
base_url = 'https://www.ncei.noaa.gov/thredds/catalog/model-narr-a-files/'

# Programmatically generate the URL to the day of data we want
cat = TDSCatalog(f'{base_url}{dt:%Y%m}/{dt:%Y%m%d}/catalog.xml')

# Have Siphon find the appropriate dataset
ds = cat.datasets.filter_time_nearest(dt)

# Interface with the data through the NetCDF Subset Service (NCSS)
ncss = ds.subset()

# Create an NCSS query with our desired specifications
query = ncss.query()
query.lonlat_box(north=60, south=18, east=300, west=225)
query.all_times()
query.add_lonlat()
query.accept('netcdf')
query.variables('Temperature_surface')

# Use the query to obtain our NetCDF data
data = ncss.get_data(query)
```

If the internet connection is too slow or if the NCEI THREDDS data server is down, we keep a copy of the subset NetCDF file locally on Anvil for you to use.

```
# In case of bad internet connection.
# Uncomment the following line to read local netCDF file
#  of the NARR data by removing the # sign leading the line.
# data = Dataset('../../../DATA/Week1/NARR_19960123_0000.nc','r')
```

Let's see what dimensions are in the file:

```
data.dimensions
```

```
{'time': <class 'netCDF4._netCDF4.Dimension'>: name = 'time', size = 1,
 'y': <class 'netCDF4._netCDF4.Dimension'>: name = 'y', size = 119,
 'x': <class 'netCDF4._netCDF4.Dimension'>: name = 'x', size = 268}
```

Pulling Data for Plotting

The object that we get from Siphon is NetCDF-like, so we can pull data using familiar calls for all of the variables that are desired for calculations and plotting purposes.

NOTE: Due to the curvilinear nature of the NARR grid, there is a need to smooth the data that we import for calculation and plotting purposes.

Additionally, we want to attach units to our values for use in MetPy calculations later and it will also allow for easy conversion to other units.

```
<b>EXERCISE</b>:
Replace the 0's in the template below with your code:
 <ul>
  <li>Use the gaussian_filter function to smooth the Temperature_isobaric, Geopotential_heig
<li>Assign the units of kelvin, meter, m/s, and m/s resectively.</li>
<li>Extract the lat, lon, and isobaric1 variables.</li>
</ul>
```

```
# Extract data and assign units
# tmpk = data.variables['Temperature_surface'][0]* units.K
tmpk = gaussian_filter(data.variables['Temperature_surface'][0],
                       sigma=1.0) * units.K
# Extract coordinate data for plotting
lat = data.variables['lat'][:]
lon = data.variables['lon'][:]
lev = 0
```

Next we need to extract the time variable. It's not in very useful units, but the `num2date` function can be used to easily create regular datetime objects.

```
time = data.variables['time']
print(time.units)
vtime = num2date(time[0], units=time.units)
print(vtime)
```

```
Hour since 1996-01-24T00:00:00Z
1996-01-24 00:00:00
```

Finally, we need to calculate the spacing of the grid in distance units instead of degrees using the MetPy helper function `lat_lon_grid_spacing`.

```
# Calcualte dx and dy for calculations
dx, dy = mpcalc.lat_lon_grid_deltas(lon, lat)
```

Maps and Projections

```
# Data projection; NARR Data is Earth Relative
dataproj = ccrs.PlateCarree()

# Plot projection
# The look you want for the view, Lambert Conformal for mid-latitude view
plotproj = ccrs.LambertConformal(central_longitude=-100., central_latitude=40.,
                                 standard_parallels=[30, 60])
```

```
def create_map_background():
    fig=plt.figure(figsize=(14, 12))
    ax=plt.subplot(111, projection=plotproj)
    ax.set_extent([-125, -73, 25, 50],ccrs.PlateCarree())
    ax.coastlines('50m', linewidth=0.75)
    ax.add_feature(cfeature.STATES, linewidth=0.5)
    return fig, ax
```

```
fig, ax = create_map_background()

# Contour 1 - Temperature, dotted
cs2 = ax.contour(lon, lat, tmpk.to('degC'), range(-50, 50, 5),
                 colors='grey', linestyles='dotted', transform=dataproj)

plt.clabel(cs2, fontsize=10, inline=1, inline_spacing=10, fmt='%i',
           rightside_up=True, use_clabeltext=True)

# Filled contours - Temperature advection
contours = [-25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25]
cf = ax.contourf(lon, lat, tmpk.to('degC'), contours,
                 cmap='bwr', extend='both', transform=dataproj)
plt.colorbar(cf, orientation='horizontal', pad=0, aspect=50,
             extendrect=True, ticks=contours)

# Titles
plt.title('Surface Temperature', loc='left')
```

```
plt.title(f'VALID: {vtime}', loc='right')

plt.show()
```