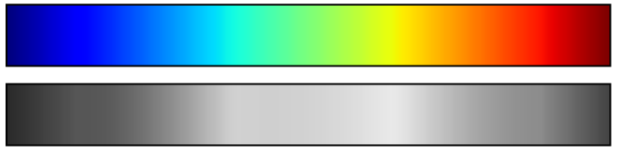
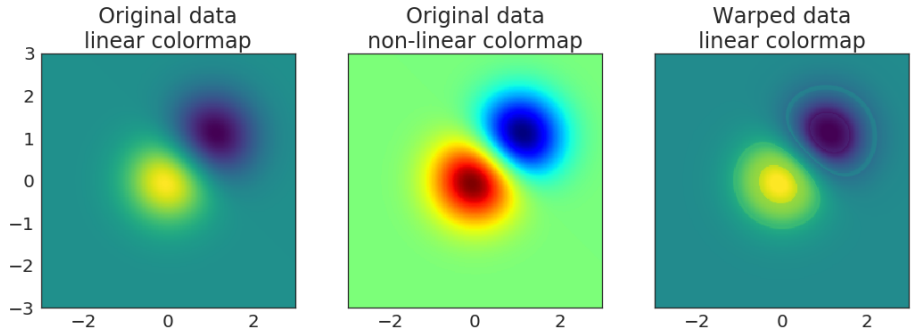
Stay away from the jet (rainbow color map)

Luminance spikes in center



http://jakevdp.github.io/blog/2014/10/16/how-bad-is-your-colormap/

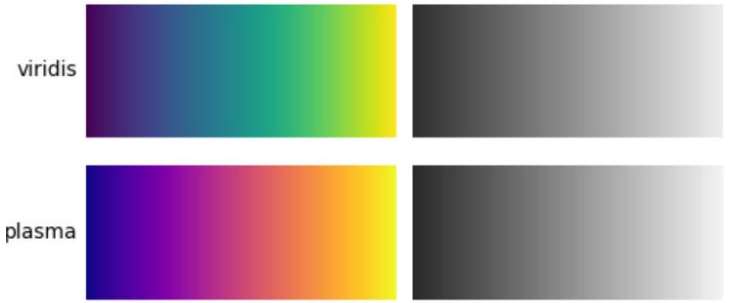
Warps your perception of the visualized data, seeing non-linearities when there aren't any



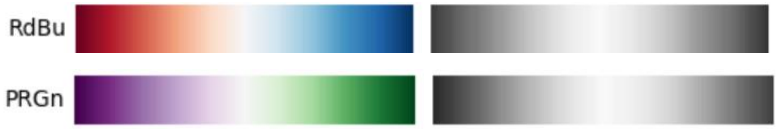
https://predictablynoisy.com/makeitpop-intro

Instead, perhaps use these!

Sequential:
virdis or plasma



Diverging:
RdBu_r (reversed) or PrGn_r



Catgorical:
Accent/Dark2

Reading CSV

Read straight from URL! So cool!

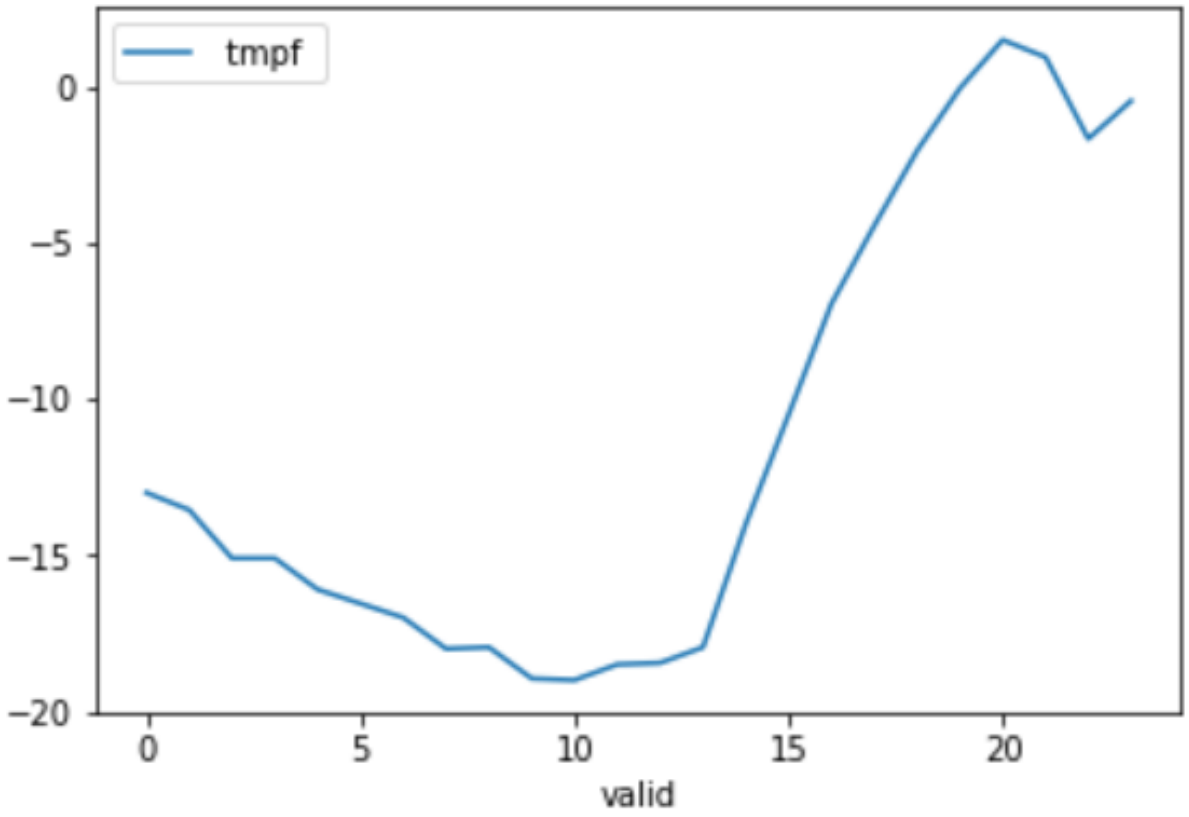
```
import pandas as pd

url = 'https://mesonet.agron.iastate.edu/cgi-bin/request/asos
df = pd.read_csv(url, index_col='valid', parse_dates=True)
print(df.head())  # print first 5
```

```
                     station  tmpf
valid
2018-01-01 00:00:00     ALO     M
2018-01-01 00:05:00     ALO     M
2018-01-01 00:10:00     ALO     M
2018-01-01 00:15:00     ALO     M
2018-01-01 00:20:00     ALO     M
```

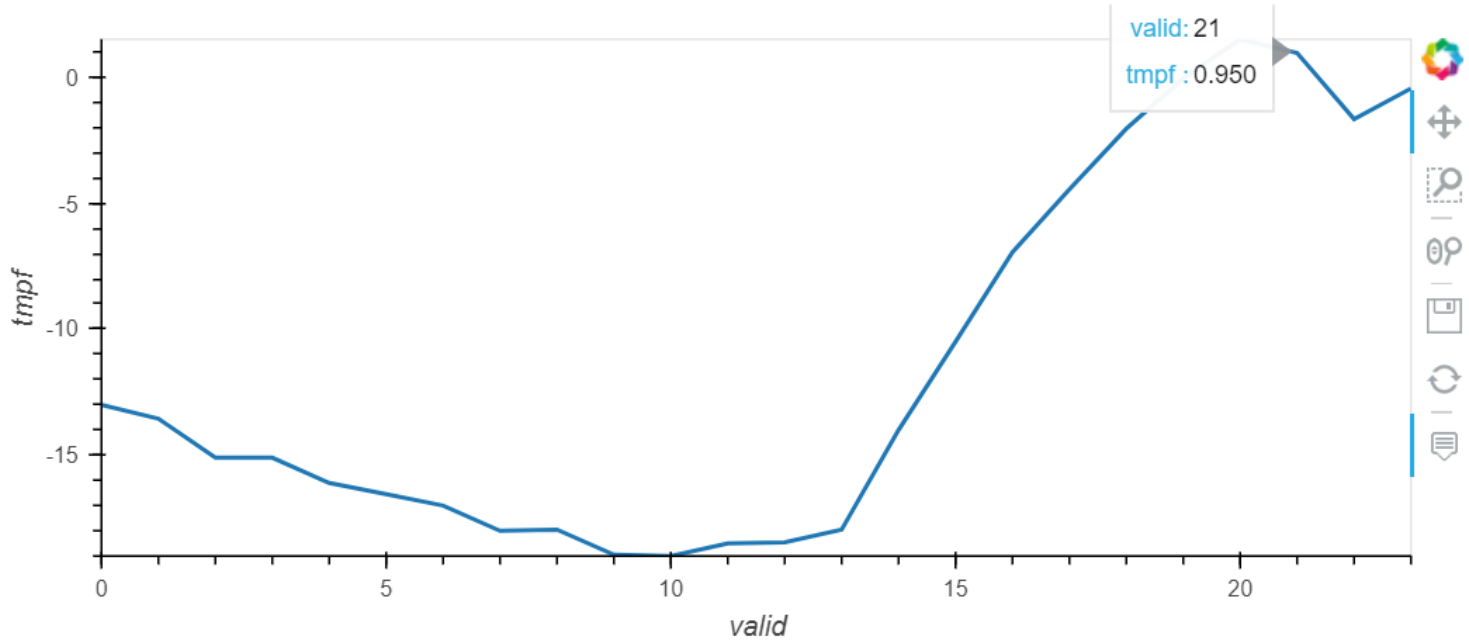
Plot diurnal cycle

```
df.groupby(df.index.hour).mean().plot()
```



Get an interactive plot

```
import holoplot.pandas
df.groupby(df.index.hour).mean().holoplot()
```

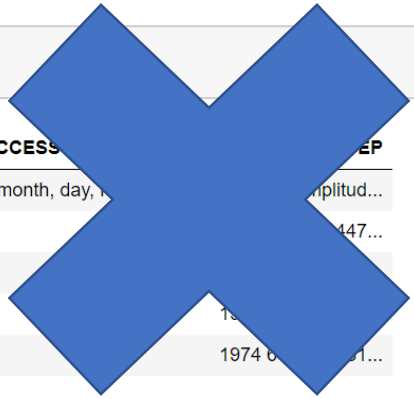


Pandas can also read unformatted tables!

```
RMM values up to "real time". For the last few days, ACCESS analyses are used instead of NCEP
year, month, day, RMM1, RMM2, phase, amplitude.  Missing Value= 1.E36 or 999
      1974          6          1   1.6344700      1.2030400              5   2.0294800      Final_value:__OLR_&_NCEP_winds
      1974          6          2   1.6028900      1.0151200              5   1.8972900      Final_value:__OLR_&_NCEP_winds
      1974          6          3   1.5162500      1.0855100              5   1.8647600      Final_value:__OLR_&_NCEP_winds
      1974          6          4   1.5098100      1.0357300              5   1.8309200      Final_value:__OLR_&_NCEP_winds
      1974          6          5   1.5590600      1.3051800              5   2.0332601      Final_value:__OLR_&_NCEP_winds
      1974          6          6   1.2062600      1.6288900              6   2.0269001      Final_value:__OLR_&_NCEP_winds
      1974          6          7  0.61110097      1.7224801              6   1.8276700      Final_value:__OLR_&_NCEP_winds
      1974          6          8  0.32639500      1.7781800              6   1.8078901      Final_value:__OLR_&_NCEP_winds
```

```
df = pd.read_table('rmm.74toRealtime.txt')
df.head()
```

	RMM values up to "real time". For the last few days, ACCESS	EP
0	year, month, day,	mplitud...
1		447...
2		
3		
4	1974	1...



```
year, month, day, RMM1, RMM2, phase, amplitude, Missing Value= 1.E36 or 999
1974      6      1   1.6344700   1.2030400      5   2.0294800  Final_value:__OLR_&_NCEP_winds
1974      6      2   1.6028900   1.0151200      5   1.8972900  Final_value:__OLR_&_NCEP_winds
1974      6      3   1.5162500   1.0855100      5   1.8647600  Final_value:__OLR_&_NCEP_winds
1974      6      4   1.5098100   1.0357300      5   1.8309200  Final_value:__OLR_&_NCEP_winds
1974      6      5   1.5590600   1.3051800      5   2.0332601  Final_value:__OLR_&_NCEP_winds
1974      6      6   1.2062600   1.6288900      6   2.0269001  Final_value:__OLR_&_NCEP_winds
1974      6      7  0.61110097   1.7224801      6   1.8276700  Final_value:__OLR_&_NCEP_winds
1974      6      8  0.32639500   1.7781800      6   1.8078901  Final_value:__OLR_&_NCEP_winds
```

```
df = pd.read_csv('rmm.74toRealtime.txt',
                 skiprows=1,
                 delimiter='\s+',
                 parse_dates=[['year,', 'month,', 'day,']],
                 index_col='year,_month,_day,')
df.columns = df.columns.str.replace(',', '').str.replace('.', '')
df = df[df.columns[:4]]
df.index.name = 'datetime'
df.head()
```

datetime	RMM1	RMM2	phase	amplitude
1974-06-01	1.63447	1.20304	5	2.02948
1974-06-02	1.60289	1.01512	5	1.89729
1974-06-03	1.51625	1.08551	5	1.86476
1974-06-04	1.50981	1.03573	5	1.83092
1974-06-05	1.55906	1.30518	5	2.03326

```
year, month, day, RMM1, RMM2, phase, amplitude.  Missing Value= 1.E36 or 999
      1974      6      1   1.6344700      1.2030400      5   2.0294800      Final_value:__OLR_&_NCEP_winds
      1974      6      2   1.6028900      1.0151200      5   1.8972900      Final_value:__OLR_&_NCEP_winds
      1974      6      3   1.5162500      1.0855100      5   1.8647600      Final_value:__OLR_&_NCEP_winds
      1974      6      4   1.5098100      1.0357300      5   1.8309200      Final_value:__OLR_&_NCEP_winds
      1974      6      5   1.5590600      1.3051800      5   2.0332601      Final_value:__OLR_&_NCEP_winds
      1974      6      6   1.2062600      1.6288900      6   2.0269001      Final_value:__OLR_&_NCEP_winds
      1974      6      7  0.61110097      1.7224801      6   1.8276700      Final_value:__OLR_&_NCEP_winds
      1974      6      8  0.32639500      1.7781800      6   1.8078901      Final_value:__OLR_&_NCEP_winds
```

```
df = pd.read_csv('rmm.74toRealtime.txt',
                 skiprows=1,
                 delimiter='\s+',
                 parse_dates=[['year,', 'month,', 'day,']],
                 index_col='year,_month,_day,')
df.columns = df.columns.str.replace(',', '').str.replace('.', '')
df = df[df.columns[:4]]
df.index.name = 'datetime'
df.head()
```

datetime	RMM1	RMM2	phase	amplitude
1974-06-01	1.63447	1.20304	5	2.02948
1974-06-02	1.60289	1.01512	5	1.89729
1974-06-03	1.51625	1.08551	5	1.86476
1974-06-04	1.50981	1.03573	5	1.83092
1974-06-05	1.55906	1.30518	5	2.03326

```
year, month, day, RMM1, RMM2, phase, amplitude,  Missing Value= 1.E36 or 999
      1974      6      1   1.6344700      1.2030400              5   2.0294800      Final_value:__OLR_&_NCEP_winds
      1974      6      2   1.6028900      1.0151200              5   1.8972900      Final_value:__OLR_&_NCEP_winds
      1974      6      3   1.5162500      1.0855100              5   1.8647600      Final_value:__OLR_&_NCEP_winds
      1974      6      4   1.5098100      1.0357300              5   1.8309200      Final_value:__OLR_&_NCEP_winds
      1974      6      5   1.5590600      1.3051800              5   2.0332601      Final_value:__OLR_&_NCEP_winds
      1974      6      6   1.2062600      1.6288900              6   2.0269001      Final_value:__OLR_&_NCEP_winds
      1974      6      7  0.61110097      1.7224801              6   1.8276700      Final_value:__OLR_&_NCEP_winds
      1974      6      8  0.32639500      1.7781800              6   1.8078901      Final_value:__OLR_&_NCEP_winds
```

```
df = pd.read_csv('rmm.74toRealtime.txt',
                 skiprows=1,
                 delimiter='\s+',
                 parse_dates=[['year,', 'month,', 'day,']],
                 index_col='year,_month,_day,')
df.columns = df.columns.str.replace(',', '').str.replace('.', '')
df = df[df.columns[:4]]
df.index.name = 'datetime'
df.head()
```

datetime	RMM1	RMM2	phase	amplitude
1974-06-01	1.63447	1.20304	5	2.02948
1974-06-02	1.60289	1.01512	5	1.89729
1974-06-03	1.51625	1.08551	5	1.86476
1974-06-04	1.50981	1.03573	5	1.83092
1974-06-05	1.55906	1.30518	5	2.03326

```
RMM values up to "real time". For the last few days, ACCESS analyses are used instead of NCEP
year, month, day, RMM1, RMM2, phase, amplitude.  Missing Value= 1.E36 or 999
       1974           6           1   1.6344700       1.2030400               5   2.0294800      Final_value:__OLR_&_NCEP_winds
       1974           6           2   1.6028900       1.0151200               5   1.8972900      Final_value:__OLR_&_NCEP_winds
       1974           6           3   1.5162500       1.0855100               5   1.8647600      Final_value:__OLR_&_NCEP_winds
       1974           6           4   1.5098100       1.0357300               5   1.8309200      Final_value:__OLR_&_NCEP_winds
       1974           6           5   1.5590600       1.3051800               5   2.0332601      Final_value:__OLR_&_NCEP_winds
       1974           6           6   1.2062600       1.6288900               6   2.0269001      Final_value:__OLR_&_NCEP_winds
       1974           6           7  0.61110097       1.7224801               6   1.8276700      Final_value:__OLR_&_NCEP_winds
       1974           6           8  0.32639500       1.7781800               6   1.8078901      Final_value:__OLR_&_NCEP_winds
```

```python
df = pd.read_csv('rmm.74toRealtime.txt',
                 skiprows=1,
                 delimiter='\s+',
                 parse_dates=[['year,', 'month,', 'day,']],
                 index_col='year,_month,_day,')
df.columns = df.columns.str.replace(',', '').str.replace('.', '')
df = df[df.columns[:4]]
df.index.name = 'datetime'
df.head()
```

Beautiful, useable output in a just few lines with no “for loops”!

```
df = pd.read_csv('rmm.74toRealtime.txt',
                 skiprows=1,
                 delimiter='\s+',
                 parse_dates=[['year,', 'month,', 'day,']],
                 index_col='year,_month,_day,')
df.columns = df.columns.str.replace(',', '').str.replace('.', '')
df = df[df.columns[:4]]
df.index.name = 'datetime'
df.head()
```

datetime	RMM1	RMM2	phase	amplitude
1974-06-01	1.63447	1.20304	5	2.02948
1974-06-02	1.60289	1.01512	5	1.89729
1974-06-03	1.51625	1.08551	5	1.86476
1974-06-04	1.50981	1.03573	5	1.83092
1974-06-05	1.55906	1.30518	5	2.03326

Reading data with higher dimensions: netCDF

```
import xarray as xr
ds = xr.open_mfdataset('air.sig995.20*.nc')
print(ds)
```

```
<xarray.Dataset>
Dimensions:    (lat: 73, lon: 144, nbnds: 2, time: 886)
Coordinates:
  * lat        (lat) float32 90.0 87.5 85.0 82.5 80.0 77.5 75.0 72.5 70.0 ...
  * lon        (lon) float32 0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0 22.5 ...
  * time       (time) datetime64[ns] 2016-01-01 2016-01-02 2016-01-03 ...
Dimensions without coordinates: nbnds
Data variables:
    air        (time, lat, lon) float32 dask.array<shape=(886, 73, 144), chunksize=(366, 73, 144)>
    time_bnds  (time, nbnds) float64 dask.array<shape=(886, 2), chunksize=(366, 2)>
Attributes:
    Conventions:    COARDS
    title:          mean daily NMC reanalysis (2014)
    history:        created 2013/12 by Hoop (netCDF2.3)
    description:    Data is from NMC initialized reanalysis\n(4x/day).  These...
    platform:       Model
    References:     http://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reana...
    dataset_title:  NCEP-NCAR Reanalysis 1
```

Get the monthly “climatology” (only two years here)

```
ds_air = ds['air']
ds_air_avg = ds_air.groupby('time.month').mean('time') # two year monthly mean (ten plus years = monthly climatology)
ds_air_avg
```

```
<xarray.DataArray 'air' (month: 12, lat: 73, lon: 144)>
dask.array<shape=(12, 73, 144), dtype=float32, chunksize=(1, 73, 144)>
Coordinates:
  * lat      (lat) float32 90.0 87.5 85.0 82.5 80.0 77.5 75.0 72.5 70.0 67.5 ...
  * lon      (lon) float32 0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0 22.5 ...
  * month    (month) int64 1 2 3 4 5 6 7 8 9 10 11 12
```

Similar to NCL’s clmMonTLL

Get the daily anomalies

```
ds_air_anom = ds_air.groupby('time.month') - ds_air_avg
```

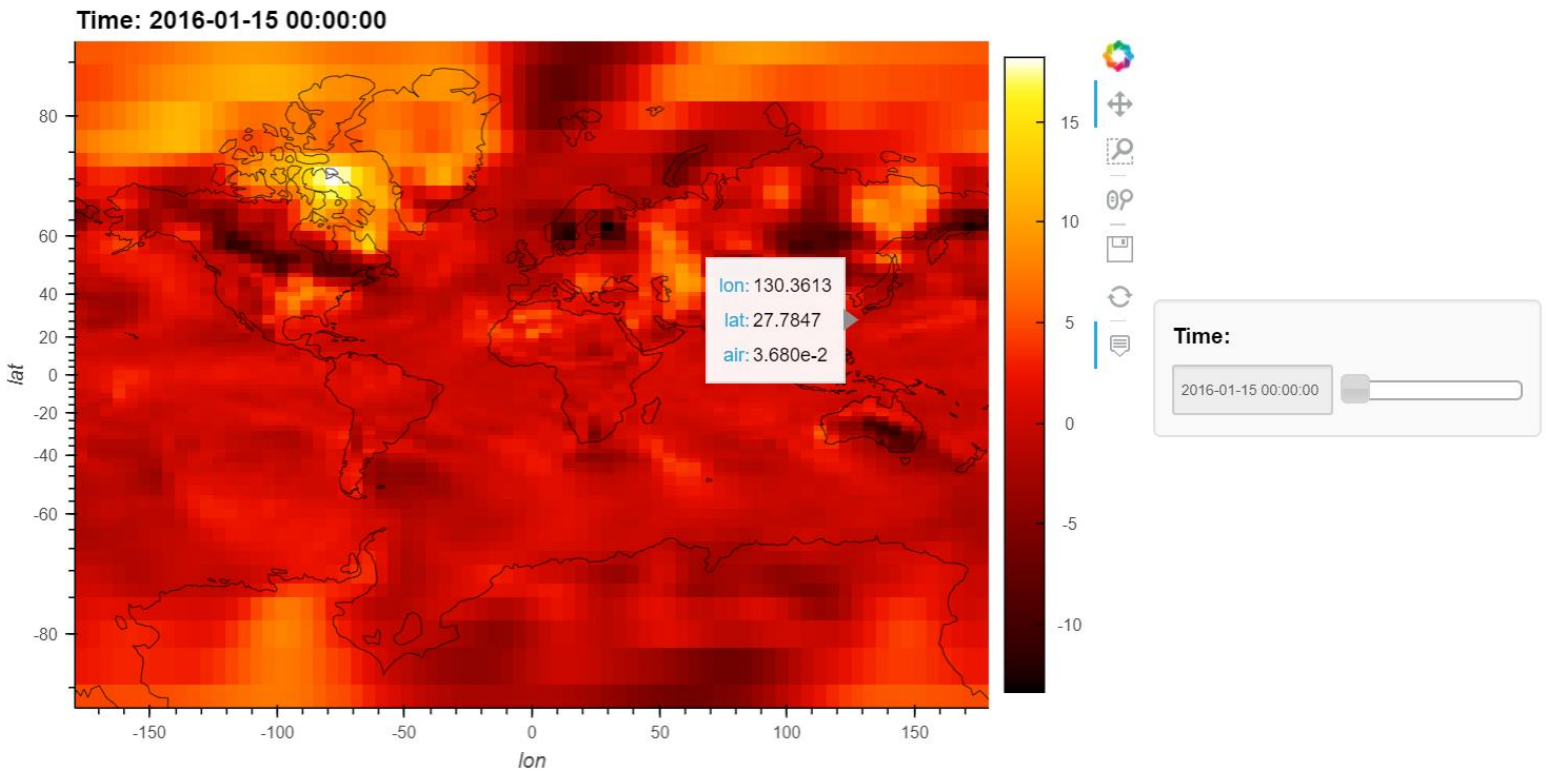
Similar to NCL’s calcMonAnomTLL

Get interactive map easily!

In [17]:
```
import geoviews as gv
import holoplot.xarray

air = ds_air_anom.holoplot('lon', 'lat',
                           groupby='time',
                           geo=True,
                           width=725,
                           height=500) * gv.feature.coastline()
air
```

Out[17]:



Reading GRIB

```
import xarray as xr

ds = xr.open_dataset('prate.2018050100.01.CFSv2.anom.avrg.1x1.grb', engine='pynio')
print(ds)
```

```
<xarray.Dataset>
Dimensions:            (forecast_time0: 8, lat_3: 181, lon_3: 360)
Coordinates:
  * lat_3              (lat_3) float32 90.0 89.0 88.0 87.0 86.0 85.0 84.0 ...
  * lon_3              (lon_3) float32 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 ...
  * forecast_time0     (forecast_time0) int32 2 3 4 5 6 7 8 9
Data variables:
    PRATE_3_SFC_ave1m  (forecast_time0, lat_3, lon_3) float32 ...
```

But those data has metadata! What about models that output plain binary?

```
import numpy as np

ntime = 1
nlat = 181
nlon = 360
fn = 'grdtmp_sfc_0000.0_0000.0_glob360x181_2017082012_01020000_fcstfld'
data = np.fromfile(fn, dtype='>f').reshape((ntime, nlat, nlon))
print(data)
```

```
[[[217.52127 217.52124 217.52122 ... 217.52129 217.52129 217.52129]
  [216.6976  216.66498 216.67284 ... 216.69624 216.68231 216.67058]
  [215.18414 215.0505  215.03452 ... 215.38925 215.3056  215.22244]
  ...
  [271.52542 271.51984 271.493   ... 271.4726  271.44864 271.48325]
  [271.51227 271.50754 271.49915 ... 271.5336  271.5274  271.51855]
  [272.10092 272.10092 272.10092 ... 272.10092 272.10092 272.10092]]]
```

Processing plain binary

Make xr.DataArray

```
import pandas as pd

LAT_RANGE = (-90, 90)
LON_RANGE = (0, 360)

dlon = LON_RANGE[1] / nlon
lon = np.arange(LON_RANGE[0], LON_RANGE[1], dlon)

dlat = (LAT_RANGE[1] - LAT_RANGE[0]) / (nlat - 1)
lat = np.arange(LAT_RANGE[0], LAT_RANGE[1] + dlat, dlat)

time = pd.to_datetime(fn.split('_')[5], format='%Y%m%d%H')

ds = xr.DataArray(data,
                  coords={'time': [time], 'lat': lat, 'lon': lon},
                  dims=('time', 'lat', 'lon'))
print(ds)
```

```
<xarray.DataArray (time: 1, lat: 181, lon: 360)>
array([[[217.52127, 217.52124, ..., 217.52129, 217.52129],
        [216.6976 , 216.66498, ..., 216.68231, 216.67058],
        ...,
        [271.51227, 271.50754, ..., 271.5274 , 271.51855],
        [272.10092, 272.10092, ..., 272.10092, 272.10092]]], dtype=float32)
Coordinates:
  * time     (time) datetime64[ns] 2017-08-20T12:00:00
  * lat      (lat) float64 -90.0 -89.0 -88.0 -87.0 -86.0 -85.0 -84.0 -83.0 ...
  * lon      (lon) float64 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
```

Reading Shapefiles

```
import geopandas as gpd

gdf = gpd.read_file('cb_2017_us_county_500k.shp')
gdf.head()
```

	STATEFP	COUNTYFP	COUNTYNS	AFFGEOID	GEOID	NAME	LSAD	ALAND	AWATER	
0	01	005	00161528	0500000US01005	01005	Barbour	06	2292144656	50538698	POLYGON ((-
1	01	023	00161537	0500000US01023	01023	Choctaw	06	2365869837	19144469	POLYGON ((-
2	01	035	00161543	0500000US01035	01035	Conecuh	06	2201948618	6643480	POLYGON ((-
3	01	051	00161551	0500000US01051	01051	Elmore	06	1601762124	99965171	POLYGON ((-
4	01	065	00161558	0500000US01065	01065	Hale	06	1667907107	32423356	POLYGON (

Processing Shapefiles

Easily figure out the top 5 states with the highest population/sq mi

```
gdf['pop_per_mi_sq'] = (gdf['DP0010001'] / gdf['ALAND10'] / 3.86102e-7)  # population per square mile
gdf[['NAME10', 'pop_per_mi_sq']].sort_values('pop_per_mi_sq', ascending=False).head()  # sort and keep top 5
```

	NAME10	pop_per_mi_sq
46	District of Columbia	9856.491855
23	New Jersey	1195.490197
7	Puerto Rico	1088.211268
5	Rhode Island	1018.139860
30	Massachusetts	839.433705

HoloViews

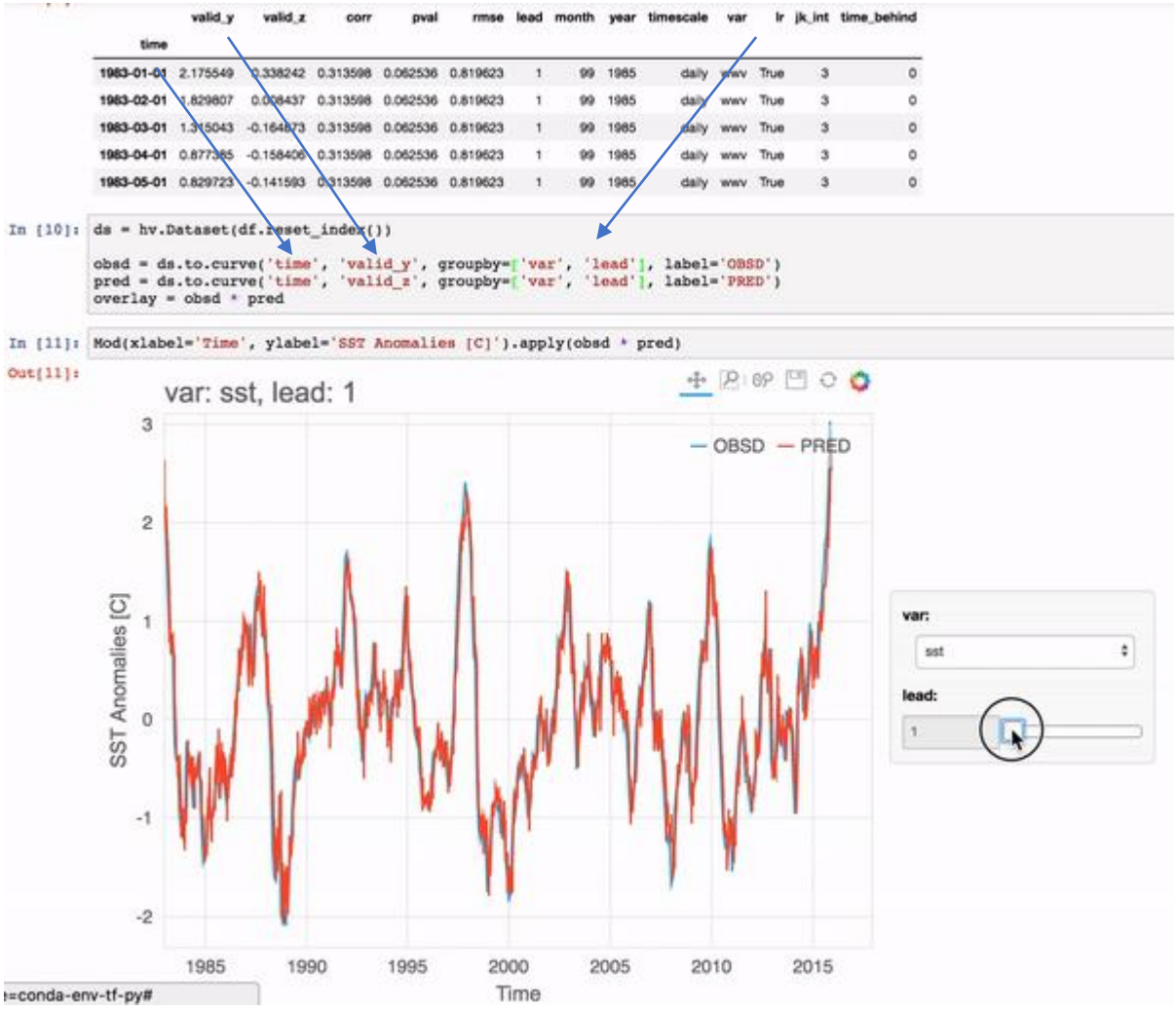
Have a DataFrame

Wrap hv.Dataset

Convert to curve by providing column names to x, y, groupby

Multiply two curves to stack them

Have awesome clickable animations!



Cartopy

```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

fig = plt.figure(figsize=(12, 8))
projection = ccrs.PlateCarree()
ax = plt.axes(projection=projection)
im = ax.contourf(da.lon, da.lat, da[0].values, transform=projection, cmap='RdBu_r')
_ = plt.colorbar(im, orientation='horizontal', shrink=0.5, pad=0.05)
_ = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
_ = ax.coastlines()
plt.grid()
```