In [135…

```
import xesmf as xe
from open_radar_data import DATASETS
import xradar as xd
import xarray as xr
import numpy as np
import cmweather
```

In [121…

```
file = '/Users/syed44/Library/Caches/open-radar-data/swx_20120520_0641.nc'
dtree = xd.io.open_cfradial1_datatree(file)
dtree = dtree.xradar.georeference()
```

In [250…

```
def get_cappi(dtree, height=2e3, tolerance=500,
              prefilter=True, vel_name='velocity', ref_name='reflectivity_horizontal'):
    '''
    Create CAPPI
    ------------
    '''
    # Function to get CAPPI for a specific height with tolerance
    def _process_height(ds, height=height, tolerance=tolerance):
        ds = ds.xradar.georeference()
        ds = ds.where((ds.z >= height - tolerance) &
                      (ds.z <= height + tolerance), drop=False)
        return ds

    def _prefilter(ds, vel_name, ref_name):
        vel_texture = ds[vel_name].rolling(range=50, min_periods=1, center=True).std()
        ds = ds.where(vel_texture < 50)
        ds = ds.where((ds[ref_name]>=-10) & (ds[ref_name]<75))
        return ds

    # Apply the CAPPI function over all sweeps
    cappi_tree = dtree.xradar.map_over_sweeps(_process_height)

    sweeps = []

    # Reference azimuth and range for alignment
    reference_ds = cappi_tree['sweep_0'].to_dataset()
    reference_range = reference_ds.range
    reference_azimuth = reference_ds.azimuth
    start_time = reference_ds.time

    # Iterate over sweeps in the DataTree
    for swp in cappi_tree.match('sweep_*'):
        ds = cappi_tree[swp].to_dataset()
        if prefilter:
            ds = ds.pipe(_prefilter, vel_name, ref_name)
        # Interpolate range and azimuth to match the reference sweep
        ds = ds.interp(range=reference_range, azimuth=reference_azimuth, method='linear')
        # Add elevation array filled with zeros
        ds['elevation'] = xr.DataArray(
            data=np.zeros_like(ds.range.values),
            dims=['range'])
        ds = ds.drop_vars(['x','y','z'])
        ds['time'] = start_time
        sweeps.append(ds)

    # Merge sweeps into a single dataset, ensuring compatibility
    ds = xr.merge(sweeps, compat='no_conflicts')
#     ds = ds.where((ds[ref_name]>=-10) & (ds[ref_name]<100))
    ds = ds.xradar.georeference()
    return ds
```

In [254…

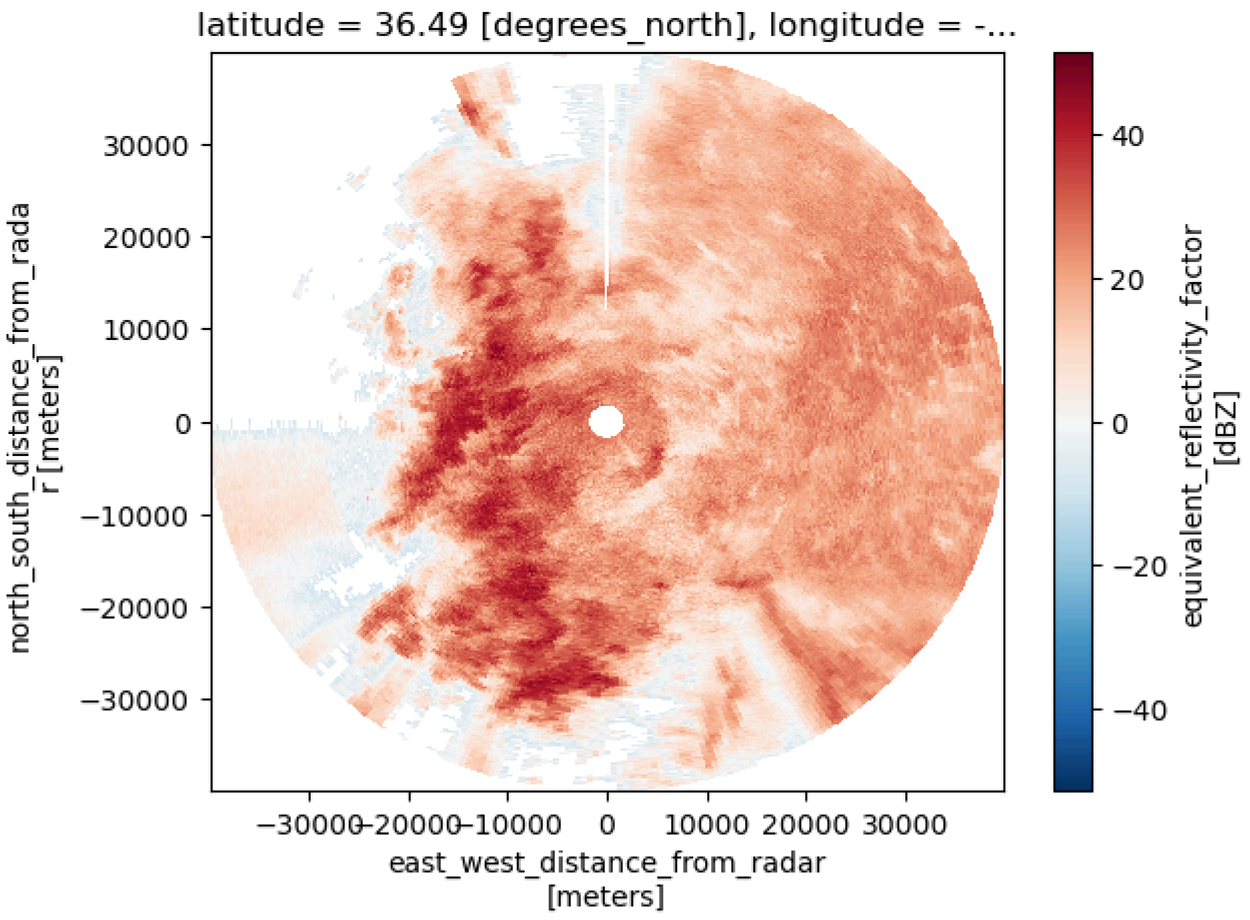
```
ds2 = get_cappi(dtree, height=2000.0, tolerance=500, prefilter=False,
                vel_name='mean_doppler_velocity', ref_name='reflectivity_horizontal')
```

In [253…

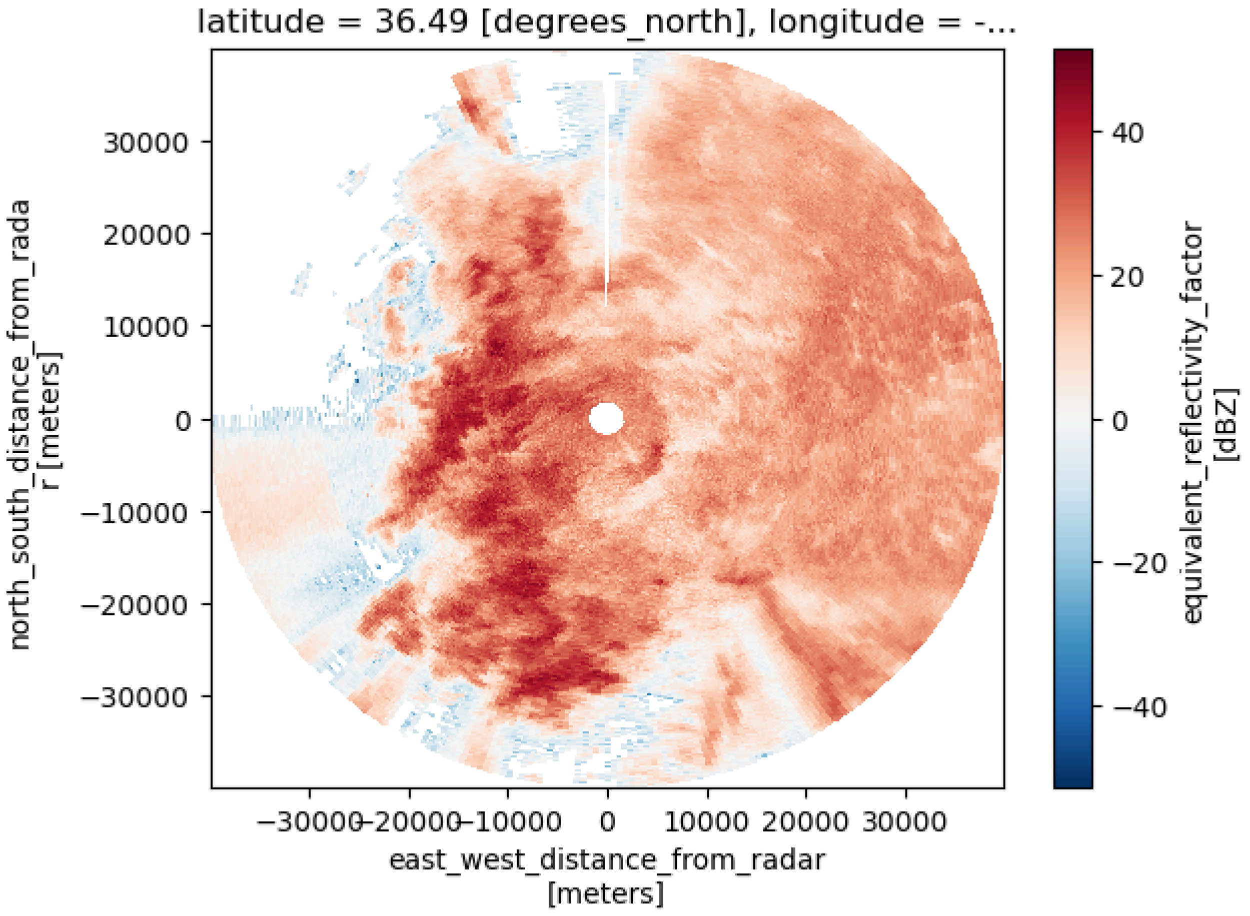
```
ds2['reflectivity_horizontal'].plot(
    x='x', y='y')
```

Out[253…

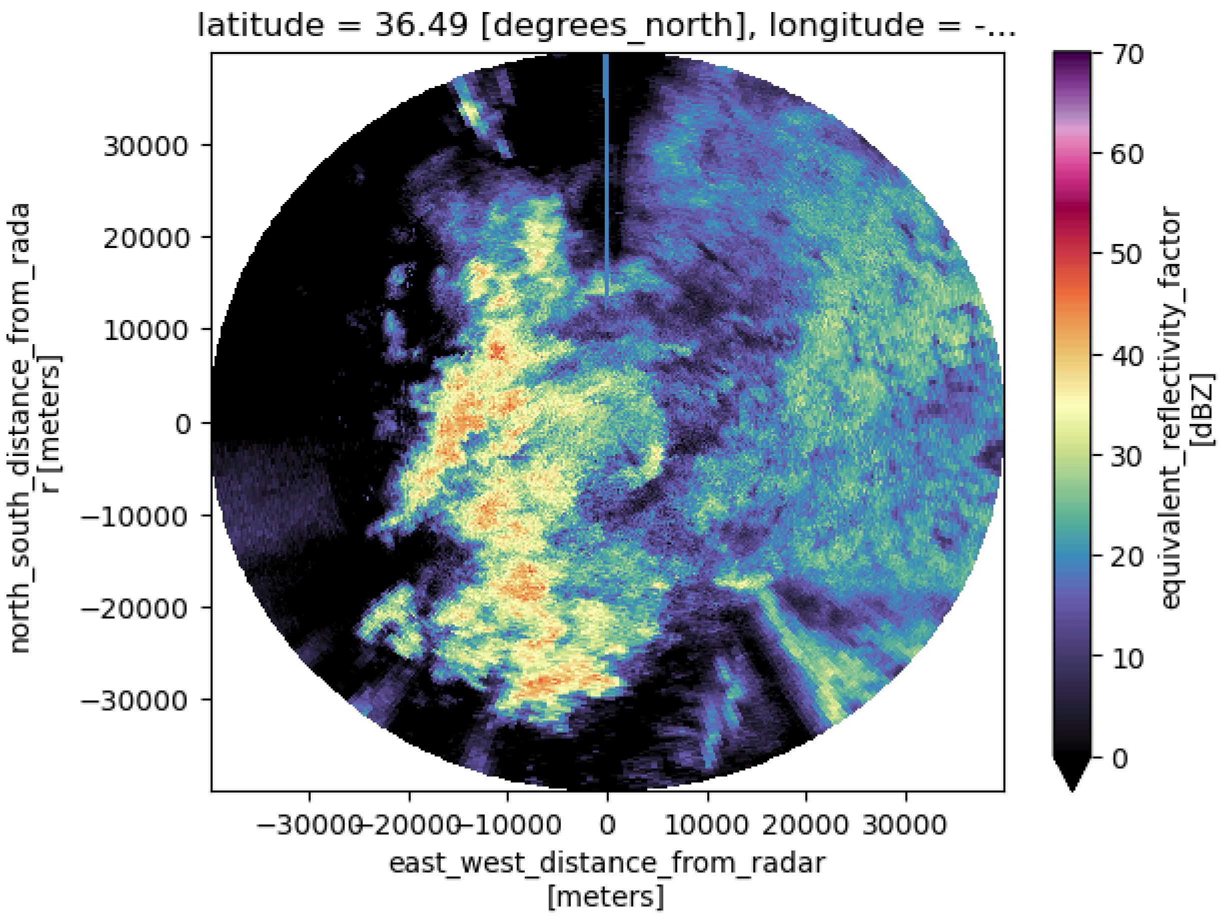
```
<matplotlib.collections.QuadMesh at 0x3e96b43e0>
```



```
In [255…  ds2['reflectivity_horizontal'].plot(
              x='x', y='y')

Out[255…  <matplotlib.collections.QuadMesh at 0x3ec84c3e0>
```



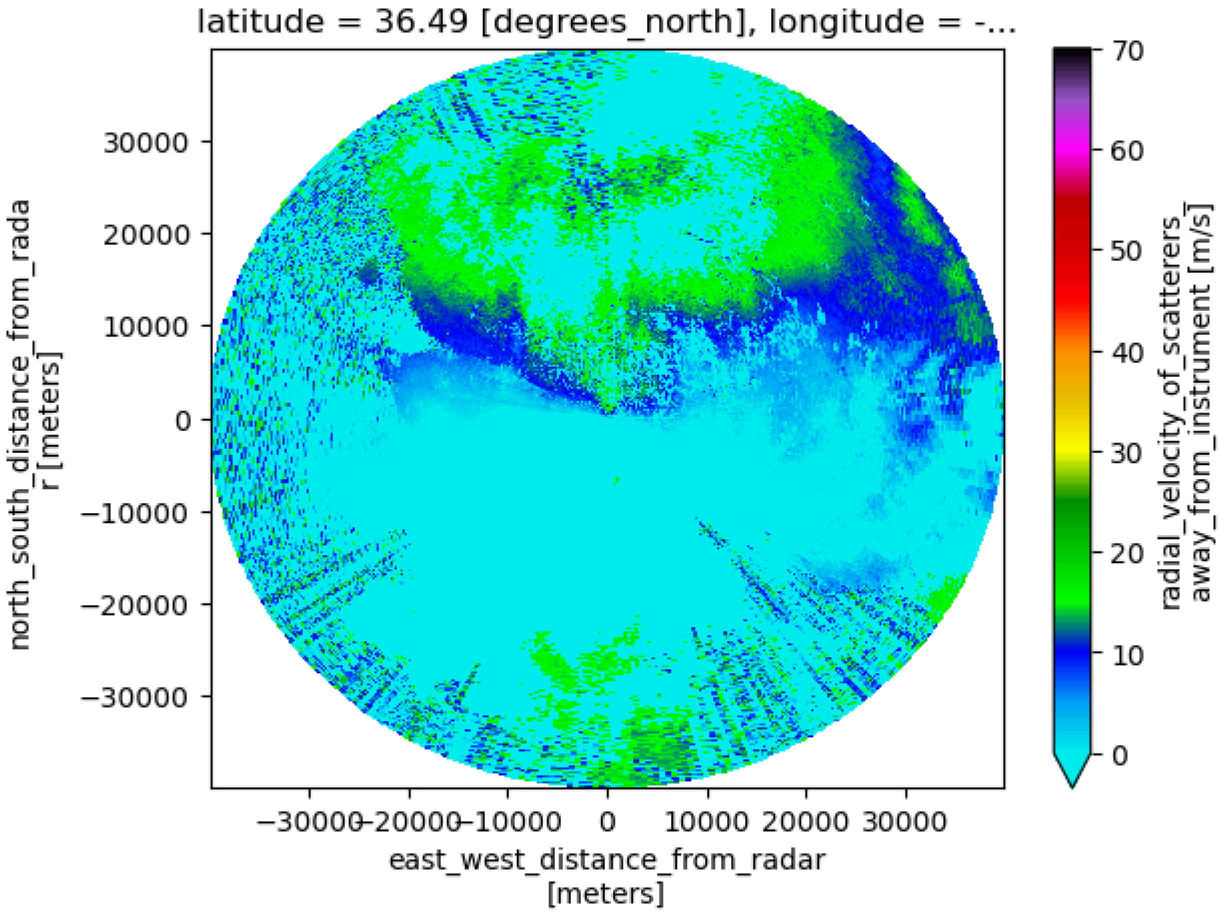
```
In [235…  ds2['reflectivity_horizontal'].interpolate_na(
              dim='range', method='nearest', fill_value='extrapolate').plot(
              x='x', y='y', vmin=0, vmax=70, cmap='ChaseSpectral')

Out[235…  <matplotlib.collections.QuadMesh at 0x3e9104410>
```



In [212…

```
ds['mean_doppler_velocity'].plot(
    x='x', y='y', vmin=0, vmax=70, cmap='NWSRef')
```

Out[212… `<matplotlib.collections.QuadMesh at 0x3d98d5e80>`

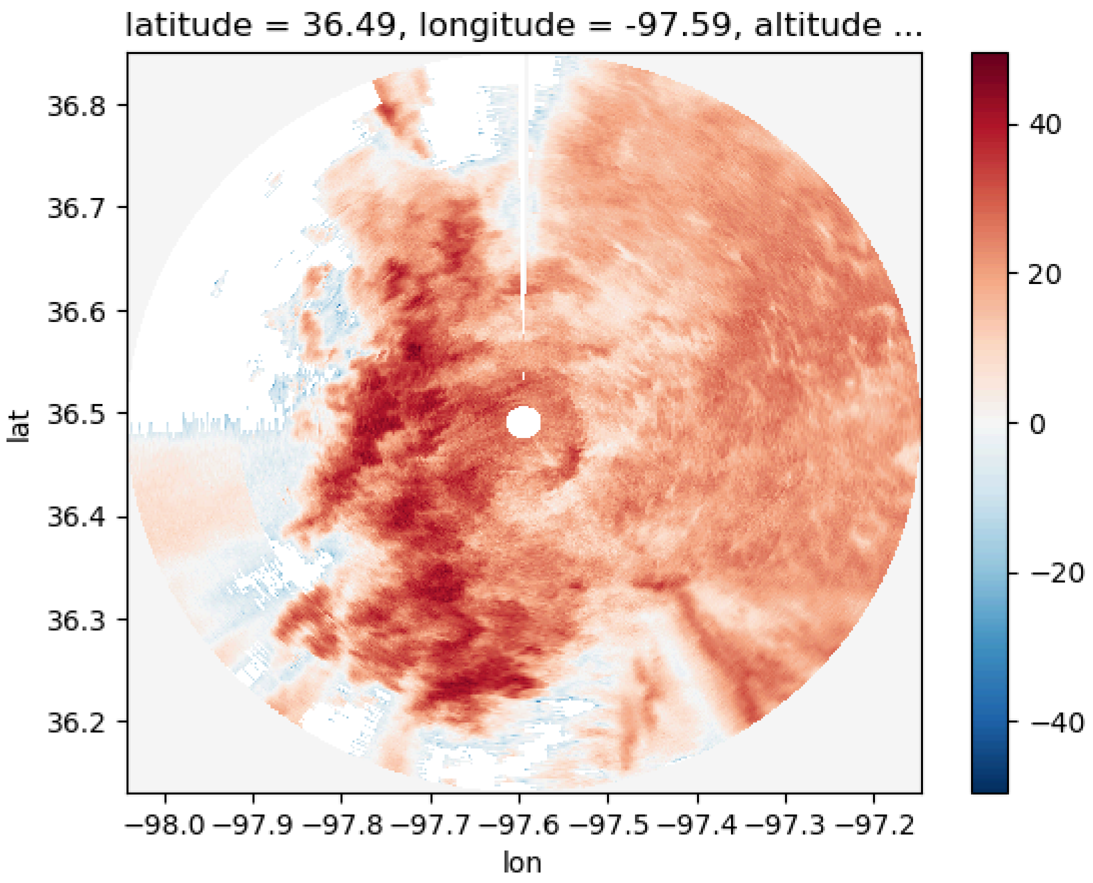


In [119…

```
# Assume radar location (lon0, lat0) is known
lon0, lat0 = ds.longitude.values, ds.latitude.values  # Example coordinates for radar site

# Convert azimuth and range to Cartesian coordinates (x, y)
R = 6371e3  # Approximate Earth radius in meters
azimuth = np.deg2rad(ds.azimuth.values)
range_vals = ds.range.values

x = range_vals[np.newaxis, :] * np.sin(azimuth[:, np.newaxis])
y = range_vals[np.newaxis, :] * np.cos(azimuth[:, np.newaxis])

# Convert Cartesian coordinates (x, y) to lat/lon
lon = lon0 + (x / R) * (180 / np.pi) / np.cos(lat0 * np.pi / 180)
lat = lat0 + (y / R) * (180 / np.pi)

# Add lon and lat to the dataset
ds['lon'] = (('azimuth', 'range'), lon)
ds['lat'] = (('azimuth', 'range'), lat)

# Create a target dataset (ds_out)
# Define a regular lat/lon grid for regridding
target_lon = np.linspace(lon.min(), lon.max(), 1000)  # Adjust resolution as needed
target_lat = np.linspace(lat.min(), lat.max(), 1000)
ds_out = xr.Dataset(
    coords={
        "lon": (["lon"], target_lon),
        "lat": (["lat"], target_lat),
    }
)

# Create the regridder
regridder = xe.Regridder(ds, ds_out, "bilinear",)

# Perform the regridding
ds_regridded = regridder(ds['reflectivity_horizontal'])

# Plot the regridded result
ds_regridded.plot(x='lon', y='lat')
```

Out[119…

```
<matplotlib.collections.QuadMesh at 0x34c0563c0>
```



In [256…

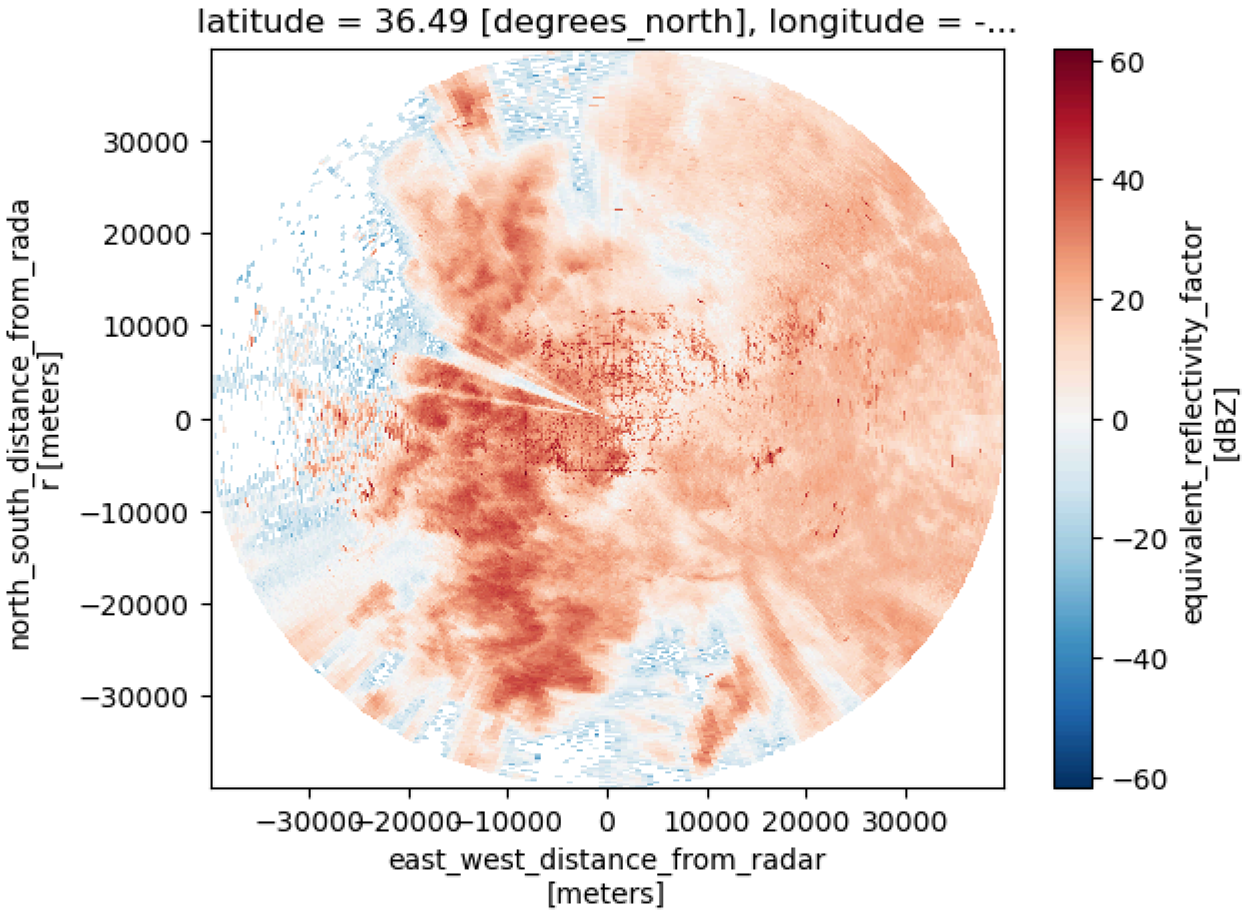
```
def _prefilter(ds, vel_name, ref_name):
    vel_texture = ds[vel_name].rolling(range=50, min_periods=1, center=True).std()
    ds = ds.where(vel_texture < 50)
    ds = ds.where((ds[ref_name]>=-10) & (ds[ref_name]<75))
    return ds
```

In [260…

```
dtree['sweep_0']['reflectivity_horizontal'].plot(x='x', y='y')
```

Out[260…

```
<matplotlib.collections.QuadMesh at 0x3ec85e480>
```

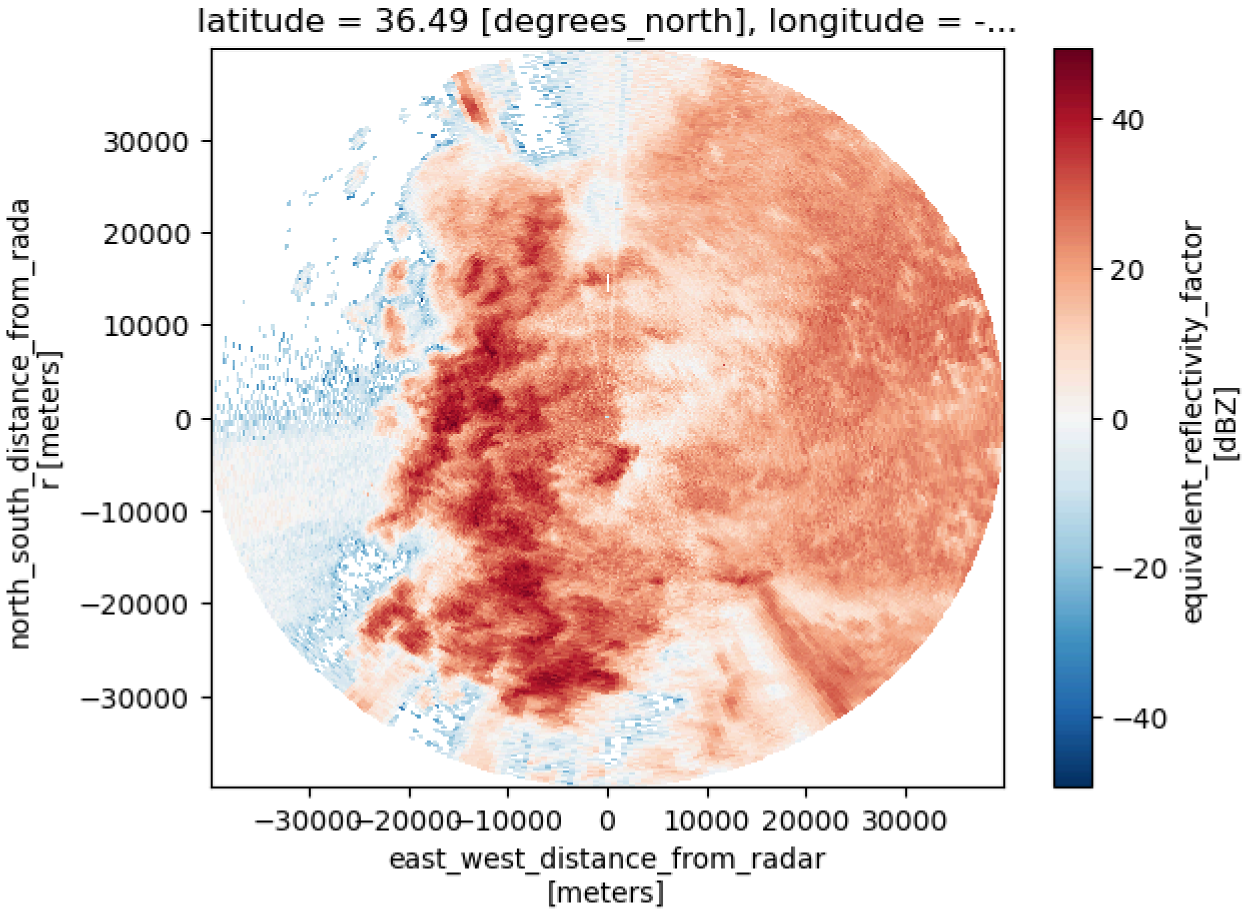


In [262…

```
dtree['sweep_3']['reflectivity_horizontal'].plot(x='x', y='y')
```

Out[262…

```
<matplotlib.collections.QuadMesh at 0x3ec6cfec0>
```



In [312…
```
ds = dtree['sweep_1'].to_dataset()
```

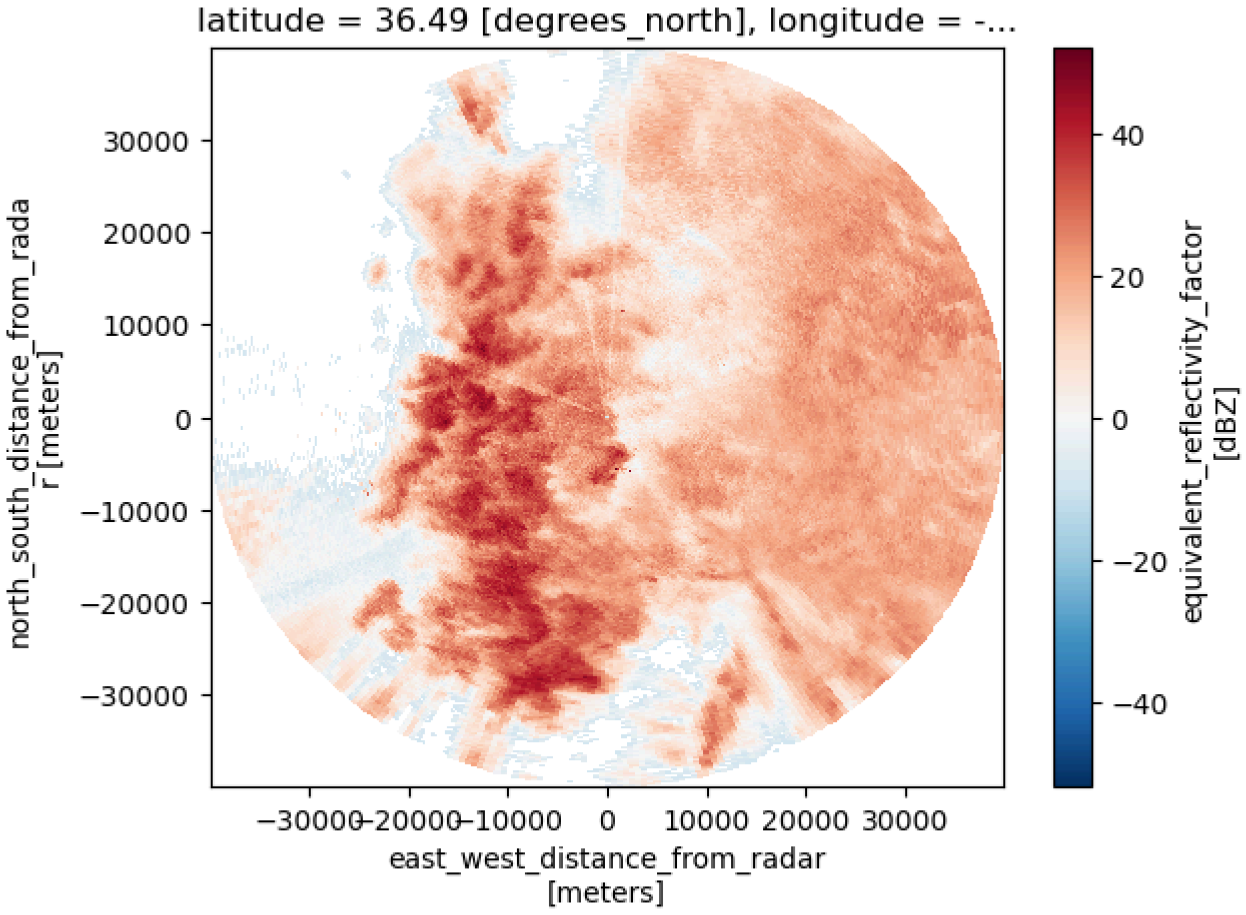
In [313…
```
import hvplot.xarray
```

In [320…
```
def prefilter(ds, vel_name, ref_name, filter_window=10):
    vel_texture = ds[vel_name].rolling(range=filter_window, min_periods=1, center=True).std()
    ds = ds.assign(velocity_texture=vel_texture)
    lim = (ds['velocity_texture'].var().item() +
           ds['velocity_texture'].std().item())
    ds = ds.where(vel_texture < abs(lim))
    ds = ds.where((ds[ref_name]>=-10) & (ds[ref_name]<75))
    return ds
```

In [321…
```
ds = ds.pipe(prefilter, 'mean_doppler_velocity', 'reflectivity_horizontal')
```

In [322…
```
ds['reflectivity_horizontal'].plot(x='x', y='y')
```

Out[322…
```
<matplotlib.collections.QuadMesh at 0x443824b90>
```



In [323…
```
ds['velocity_texture'].hvplot.density()
```

Out[323…

In [324…
```
!open .
```

In [ ]:

In [ ]: