Generate GeoTIFF Files for Meteorological Data from the GFS Model

This script allows you to generate GeoTIFF files containing meteorological data sourced from various models and forecast periods. It utilizes the Herbie library to access and download weather data in GRIB2 format and then converts it into GeoTIFF files for easy visualization and analysis.

How to Use:

1. **Installation:** Before using the script, ensure you have the required dependencies installed. You can install them via pip:

```
pip install herbie-data xarray numpy rasterio
```

2. **Running the Script:** You can run the script in any Python environment, such as a terminal or an Integrated Development Environment (IDE). In this class we often use the Python Interactive Terminal that comes with the ArcGIS download.

3. **Input Parameters:**
   - When prompted, select the period for which you want the meteorological data: "past", "current", or "forecast".
     - For the "past" period:
       - Enter the date of the event you want the data from in the format YYYY-MM-DD.
       - Specify the model run time in UTC (00, 06, 12, or 18).
     - For the "current" period, the script will automatically fetch the latest available data.
     - For the "forecast" period:
       - Enter the lead time in hours (between 1 and 384).

4. **Output:** The script will generate a GeoTIFF file containing the GFS model data for the selected period. The file will be saved in your current directory with a descriptive name indicating the data source and period. I specifically made sure the past, current, and forecast GeoTiff files have distinctly different file names.

5. **Interpreting the GeoTIFF:**

- Each band in the GeoTIFF corresponds to a different meteorological variable (listed in the ‘band table’ on the GitHub repository.

- Band names are customized to provide meaningful descriptions of each variable.
- You can visualize the GeoTIFF using GIS software like ArcGIS or QGIS for further analysis and visualization.

Example Usage:

To generate GeoTIFF files containing past meteorological data for a specific date and model run (bold is user-input):
>>>**python GFS-2-GeoTiff.py**
>>>Would you like the meteorological data to be: past, current, or forecast? **Past**
>>>The data you want is from what date (in YYYY-MM-DD format)? **2024-05-01**
>>>What model run (in UTC time) do you want the data from (00, 06, 12, or 18)? **00**

Troubleshooting:

- Ensure you have a stable internet connection to download meteorological data from Herbie.
- Double-check input parameters to ensure they are entered correctly.
- If the script encounters errors, it provides informative messages to guide troubleshooting.
- **If all else fails, make sure to report the issue on the GitHub so it can be fixed**

Script (also in GitHub Repo):

```
from herbie import Herbie, FastHerbie, HerbieLatest
import xarray as xr
import numpy as np
import xarray as xr
import rasterio
from rasterio.transform import from_origin


# Ask when the data should be from or for
period = input("Would you like the meteorological data to be: past, current, or forecast?")

if period == "past":

  # These lines are for if the user wants PAST data

  # Ask for the date of the event they want the data from
  date = input("The data you want is from what date (in YYYY-MM-DD format)?")

  # Ask for the model run they want for past data
  model_run = input("What model run (in UTC time) do you want the data from (00, 06, 12, or 18)?")
```

```
    date = str(date) # User must input it as YYYY-MM-DD
    model_run = str(model_run) # must be in format HH (out of 00, 06, 12, or 18)

    # Run Herbie to temperarily download the GRIB2 files for GFS data
    H = Herbie(date+' '+model_run+':00', model="gfs", fxx=0)

    # Let the user know the code is running...
    print("Thank you, our system is currently processing your request and your GeoTiff file should be downloaded to your current directory
shortly...")

    # Create xarray datasets for each common weather variable
    past_tmp = H.xarray(':TMP:2 m above ground')
    past_dpt = H.xarray(':DPT:2 m above ground')
    past_rh = H.xarray(':RH:2 m above ground')
    past_prate = H.xarray(':PRATE:surface')
    past_refc = H.xarray(':REFC:entire atmosphere')
    past_ugrd = H.xarray(':UGRD:10 m above ground')
    past_vgrd = H.xarray(':VGRD:10 m above ground')
    past_hlcy = H.xarray(':HLCY:3000-0 m above ground')
    past_absv = H.xarray(':ABSV:850 mb')


    # Define the output GeoTIFF file path
    output_file = 'GFS-Data-'+date+'-'+model_run+'z.tif'

    # Initialize a list to store datasets
    datasets = [past_tmp, past_dpt, past_rh, past_prate, past_refc, past_ugrd, past_vgrd, past_hlcy, past_absv]  # Add all your datasets
here

    # Adjust longitude values to range from -180 to 180
    for ds in datasets:
        lon_values = ds.longitude.values
        lon_values[lon_values > 180] -= 360

    # Get the spatial information from one of the datasets
    lon_values = datasets[0].longitude.values
    transform = from_origin(
        lon_values.min(),          # Minimum longitude (-180)
        lon_values.max(),          # Maximum latitude
        lon_values[1] - lon_values[0],   # Pixel size in longitude
        datasets[0].latitude.values[0] - datasets[0].latitude.values[1]  # Pixel size in latitude (usually negative)
    )

    # Define custom band names
    variable_names = ['2mAirTemp', '2mAirDewpnt', '2mAirRH', 'PrecipRate', 'CompositeRef', '10mUwind', '10mVwind', 'Helicity',
'850mbVort']  # Add custom names for each band


    # Write the datasets to a GeoTIFF file
    with rasterio.open(output_file, 'w', driver='GTiff',
                    width=datasets[0].dims['longitude'], height=datasets[0].dims['latitude'],
                    count=len(datasets), dtype='float32',
                    crs='EPSG:4326', transform=transform, names=band_names) as dst:
        for i, ds in enumerate(datasets):
            data = ds[list(ds.data_vars)[0]].values  # Get the data of the first variable in the dataset
            dst.write(data, i + 1)  # Write data to corresponding band

            # Update metadata to set custom variable name
            dst.set_band_description(i + 1, variable_names[i])


elif period == "current":

    # These lines are for if the user wants CURRENT data
    HL = HerbieLatest(model="gfs", fxx=0)

    # Let the user know the code is running...
    print("Thank you, our system is currently processing your request and your GeoTiff file should be downloaded to your current directory
shortly...")

    # Create xarray datasets for each common weather variable
    current_tmp = HL.xarray(':TMP:2 m above ground')
    current_dpt = HL.xarray(':DPT:2 m above ground')
    current_rh = HL.xarray(':RH:2 m above ground')
    current_prate = HL.xarray(':PRATE:surface')
    current_refc = HL.xarray(':REFC:entire atmosphere')
    current_ugrd = HL.xarray(':UGRD:10 m above ground')
    current_vgrd = HL.xarray(':VGRD:10 m above ground')
    current_hlcy = HL.xarray(':HLCY:3000-0 m above ground')
    current_absv = HL.xarray(':ABSV:850 mb')

    # Define the output GeoTIFF file path
    output_file = 'Current-GFS-Data.tif'

    # Initialize a list to store datasets
    datasets = [current_tmp, current_dpt, current_rh, current_prate, current_refc, current_ugrd, current_vgrd, current_hlcy, current_absv]
# Add all your datasets here

    # Adjust longitude values to range from -180 to 180
    for ds in datasets:
        lon_values = ds.longitude.values
        lon_values[lon_values > 180] -= 360

    # Get the spatial information from one of the datasets
    lon_values = datasets[0].longitude.values
    transform = from_origin(
        lon_values.min(),              # Minimum longitude (-180)
        lon_values.max(),              # Maximum latitude
        lon_values[1] - lon_values[0],   # Pixel size in longitude
        datasets[0].latitude.values[0] - datasets[0].latitude.values[1]  # Pixel size in latitude (usually negative)
    )

    # Define custom band names
    variable_names = ['2mAirTemp', '2mAirDewpnt', '2mAirRH', 'PrecipRate', 'CompositeRef', '10mUwind', '10mVwind', 'Helicity',
'850mbVort']  # Add custom names for each band


    # Write the datasets to a GeoTIFF file
    with rasterio.open(output_file, 'w', driver='GTiff',
                       width=datasets[0].dims['longitude'], height=datasets[0].dims['latitude'],
                       count=len(datasets), dtype='float32',
                       crs='EPSG:4326', transform=transform, names=band_names) as dst:
        for i, ds in enumerate(datasets):
            data = ds[list(ds.data_vars)[0]].values  # Get the data of the first variable in the dataset
            dst.write(data, i + 1)  # Write data to corresponding band

            # Update metadata to set custom variable name
            dst.set_band_description(i + 1, variable_names[i])


elif period == "forecast":

    # These lines are for if the user wants FORECAST data

    # Ask for the lead-time they want for forecast data
    fxx = input("How far out do you want the forecast (lead-time in hours, between 1 and 384)?")

    fxx = int(fxx)
    HL = HerbieLatest(model="gfs", fxx=fxx)
    # Let the user know the code is running...
    print("Thank you, our system is currently processing your request and your GeoTiff file should be downloaded to your current directory
shortly...")

    # Create xarray datasets for each common weather variable
    forecast_tmp = HL.xarray(':TMP:2 m above ground')
    forecast_dpt = HL.xarray(':DPT:2 m above ground')
    forecast_rh = HL.xarray(':RH:2 m above ground')
    forecast_prate = HL.xarray(':PRATE:surface')
    forecast_refc = HL.xarray(':REFC:entire atmosphere')
    forecast_ugrd = HL.xarray(':UGRD:10 m above ground')
    forecast_vgrd = HL.xarray(':VGRD:10 m above ground')
    forecast_hlcy = HL.xarray(':HLCY:3000-0 m above ground')
    forecast_absv = HL.xarray(':ABSV:850 mb')

    #For some reason prate has a step dimension but none of the others do, but its also just repeated values...
    # Assuming ds is your dataset with the 'step' dimension
    forecast_prate = forecast_prate.isel(step=0)  # Selecting the first step to keep, since both steps are the same values


    # Define the output GeoTIFF file path for the user
    output_file = 'GFS-'+str(fxx)+'hr-Forecast.tif'

    # Initialize a list to store the xarray datasets
    datasets = [forecast_tmp, forecast_dpt, forecast_rh, forecast_prate, forecast_refc, forecast_ugrd, forecast_vgrd, forecast_hlcy,
forecast_absv]  # Add all your datasets here

    # Adjust lon values to range from -180 to 180, since GFS is 0-359.75 for lon
    for ds in datasets:
        lon_values = ds.longitude.values
        lon_values[lon_values > 180] -= 360

    # Get the spatial information from one of the datasets
    lon_values = datasets[0].longitude.values
    transform = from_origin(
        lon_values.min(),             # Minimum longitude (should be -180)
        lon_values.max(),             # Maximum latitude (should be 180)
        lon_values[1] - lon_values[0],   # Pixel size in longitude
        datasets[0].latitude.values[0] - datasets[0].latitude.values[1]  # Pixel size in latitude
    )

    # Define custom band names
    variable_names = ['2mAirTemp', '2mAirDewpnt', '2mAirRH', 'PrecipRate', 'CompositeRef', '10mUwind', '10mVwind', 'Helicity',
'850mbVort']  # Add custom names for each band


    # Write the datasets to a GeoTIFF file
    with rasterio.open(output_file, 'w', driver='GTiff',
                    width=datasets[0].dims['longitude'], height=datasets[0].dims['latitude'],
                    count=len(datasets), dtype='float32',
                    crs='EPSG:4326', transform=transform, names=band_names) as dst:
        for i, ds in enumerate(datasets):
            data = ds[list(ds.data_vars)[0]].values  # Get the data of the first variable in the dataset
            dst.write(data, i + 1)  # Write data to corresponding band

            # Update metadata to set custom variable name
            dst.set_band_description(i + 1, variable_names[i])


else:
    print("Try running the code again and make sure to spell 'past', 'current', or 'forecast' correctly!")
```