Weather Hazards Monitoring with GOES-R Data

SatPy is a versatile library for **reading**, **manipulating**, and **visualizing** data from weather satellites. In this section, we will explore more advanced capabilities of SatPy for processing weather satellite data, enabling us to work effectively with meteorological datasets.

Loading and Visualizing Satellite Data

```
[ ] # Import the warnings library, which allows control over warning messages.
    import warnings

    # Disable all warning messages to keep the output clean.
    # This is useful in a lecture setting to avoid distracting students with non-critical warning messages
    # that might arise from library functions used in processing geostationary satellite data.
    warnings.filterwarnings('ignore')
```

```
[ ] # Import necessary components from the satpy library, which is crucial for satellite data processing.
    from satpy.scene import Scene  # Scene is a central object in Satpy used to represent and manipulate satellite data.

    # The 'find_files_and_readers' function automates the discovery of satellite data files and the appropriate reader based on the data's metadata.
    # This simplifies loading and processing satellite imagery, which is essential for analyzing geostationary satellite data.
    from satpy import find_files_and_readers
```

Searching for GOES-R L1B Data

```
[ ] # Import the os module to interact with the operating system and the glob module to find all pathnames matching a specified pattern.
    import os
    import glob

    # Define the base directory for satellite data relative to the current script's location.
    # This is crucial for handling data files in a way that remains functional across different systems and setups.
    base_dir = os.path.join('input_data', 'ABI-L1b-RadF')

    # Construct a pattern to find specific subdirectories under the base directory.
    # This pattern targets directories starting with 's20210992350171', which could be date-specific datasets from the ABI (Advanced Baseline Imager) on geostationary satellites.
    pattern = os.path.join(base_dir, 's20210992350171*')
    directories = glob.glob(pattern)  # Use glob to search for directories that match this pattern.

    # Select the first directory found that matches the pattern. This directory will be used to load satellite data.
    # It's common in data processing workflows to automate the selection of data subsets like this for analysis.
    directory = directories[0]

    # Load the satellite data using Satpy's find_files_and_readers function, specifying the directory and the reader type ('abi_l1b').
    # 'abi_l1b' refers to the reader for Level 1b data from the ABI instrument, which is essential for high-resolution weather imaging on geostationary satellites.
    fGRl1b = find_files_and_readers(base_dir=directory, reader='abi_l1b')

    # Output the result to show what files and readers are being used. This helps in debugging and understanding the data loading process.
    fGRl1b
```

Searching for GOES-R L2 CMIPC Data

```
[ ] import os
    import glob

    # Define the base directory for Level 2 satellite data relative to the current script's location.
    # This is pivotal for handling files in a way that ensures portability across different systems.
    base_dir = os.path.join('input_data', 'ABI-L2-CMIPF')

    # Construct a pattern to find specific subdirectories under the base directory.
    # Here, the pattern targets directories starting with 's20200621430', likely corresponding to specific observations or time points.
    pattern = os.path.join(base_dir, 's20200621430*')
    directories = glob.glob(pattern)  # Use glob to search for directories that match this pattern.

    # Select the first directory found that matches the pattern. This directory will be used to load the satellite data.
    # Automating the selection of specific data subsets simplifies the analysis process, especially in educational settings.
    directory = directories[0]

    # Load the satellite data using Satpy's find_files_and_readers function, specifying the directory and the reader type ('abi_l2_nc').
    # 'abi_l2_nc' indicates a reader for Level 2 data from the ABI instrument, formatted in NetCDF, which is used for detailed environmental data analysis.
    fGRl2 = find_files_and_readers(base_dir=directory, reader='abi_l2_nc')

    # Output the result to show what files and readers are being used. This is useful for verification and understanding the data handling process.
    fGRl2
```

```
[ ] # Import the datetime module to handle date and time data.
    # This is essential for processing time-stamped satellite data, allowing for precise time-based analysis and operations.
    from datetime import datetime

    # Import the glob function directly from the glob module.
    # This function is used to find all the file paths that match a specific pattern, which is crucial for automatically locating satellite data files.
    # Using glob allows for efficient and flexible file handling, especially when dealing with large datasets typically generated by geostationary satellites.
    from glob import glob
```

```
[ ] # Create a Scene object named 'scn' by providing a list of filenames obtained from 'fGRl1b'.
    # The Scene object is a core part of the Satpy library, which organizes and manages multiple data sources typically from satellite observations.
    scn = Scene(filenames=fGRl1b)

    # Retrieve the names of all datasets available in the 'scn' object, which represent various satellite data channels.
    # This information is crucial for understanding what types of data are available for analysis, such as different spectral bands and derived products.
    dataset_names = scn.all_dataset_names()

    # Output the list of dataset names. This is useful for educational purposes to show students the variety of data that can be processed
    # and to select specific datasets for further analysis in practical exercises.
    print(dataset_names)
```

```
['C01', 'C02', 'C03', 'C04', 'C05', 'C06', 'C07', 'C08', 'C09', 'C10', 'C11', 'C12', 'C13', 'C14', 'C15', 'C16']
```

```
[ ] # The magic command '%matplotlib inline' configures the Jupyter Notebook to display plots directly below the code cells.
    # This setting is essential for interactive data visualization, especially when plotting data from geostationary satellites.
    %matplotlib inline

    # Import the matplotlib.pyplot module under the alias 'plt'.
    # Matplotlib is a comprehensive library for creating static, interactive, and animated visualizations in Python.
    # It is particularly useful in satellite data analysis for plotting images, graphs, and other types of visual data representations.
    import matplotlib.pyplot as plt
```

```
[ ] # Load multiple datasets using a list comprehension to generate dataset names.
    # List comprehensions provide a concise way to create lists based on existing lists or ranges.
    # In this case, we generate names for the datasets 'C01' to 'C16', which are typical channel identifiers in satellite data.
    scn.load([f'C{x:02d}' for x in range(1, 17)])

    # Explanation of the list comprehension:
    # [f'C{x:02d}' for x in range(1, 17)] creates a list of strings from 'C01' to 'C16'.
    # 'f' before the string starts an f-string, allowing us to insert variables directly into the string.
    # '{x:02d}' formats the number 'x' as a two-digit decimal, padding with zeros if necessary.

    # The 'scn.load' function is then used to load these specific datasets into the Scene object.
    # Loading multiple channels like this is common in the analysis of satellite imagery,
    # where each channel can represent different spectral bands and contain different types of environmental information.
```

```
[ ] # The method 'available_composite_names' is called on the 'scn' object.
    # This method retrieves a list of all the composite images that can be created using the loaded data channels.
    # Composite images are made by combining multiple data channels to enhance the visualization and interpretation of satellite data.
    # This feature is particularly useful in the study of geostationary satellites, as it allows for more detailed and informative visual representations of atmospheric phenomena.

    # Retrieve and print the list of available composite names, providing a crucial insight into the data visualization capabilities.
    print(scn.available_composite_names())
```

```
['24h_microphysics', 'airmass', 'ash', 'cimss_cloud_type', 'cimss_cloud_type_raw', 'cimss_green', 'cimss_green_sunz', 'cimss_green_sunz_rayleigh', 'cimss_true_color', 'cimss_true_color
```

[ ]
```python
# Assign the dataset name 'airmass' to the variable 'rgb_im'.
# This variable naming provides clarity when referencing the dataset in multiple places,
# ensuring consistency and reducing the likelihood of errors in dataset identification.
rgb_im = 'airmass'

# Load the dataset named 'airmass' using the 'scn.load' method.
# The 'airmass' composite is particularly useful in meteorology as it combines several spectral bands to highlight features like dust, ash, and water vapor,
# making it easier to analyze atmospheric conditions from geostationary satellite data.
scn.load([rgb_im])

# Display the loaded 'airmass' dataset using 'scn.show'.
# This method visualizes the specified dataset, allowing students to see the practical application of satellite data composites
# and understand their relevance in real-world atmospheric monitoring and analysis.
scn.show(rgb_im)
```

```
# and understand their relevance in real world atmospheric monitoring and analysis.
scn.show(rgb_im)
```