SatPy - Part 3: Exploring The Geostationary Operational Environmental Satellite (GOES) – R Data

For detailed documentation, visit SatPy Documentation.

SatPy is a powerful library for **reading**, **manipulating**, and **displaying** data from remote sensors, primarily related to meteorology. It also provides the capability to **save** this data as images or in various formats.

SatPy excels at generating images with **individual channels or bands** and creating **RGB composites** directly from satellite instrument data.

The pyresample library is used for **resampling data** in different areas with specific projections or uniform grids.

Additionally, Satpy offers various **atmospheric corrections** and **visual enhancements**, either directly within Satpy or through the PySpectral and TrollImage packages.

The Geostationary Operational Environmental Satellite (GOES) – R Data

https://www.goes-r.gov/

Loading and Visualizing Satellite Data

In [ ]:
```
urls2dwn = [ 'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
  'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadF/2021/099/23/OR_ABI-L1b-RadF-M6
```

In [ ]:
```
import requests
import os

# Specify the local directory where you want to save the files.
# local_directory = input("Enter the path to the download folder: ")
local_directory = "Output_data/ABI-L1b-RadF/s20210992350171"
# Ensure that the local directory exists; create it if it doesn't.
os.makedirs(local_directory, exist_ok=True)

# Iterate through the URLs and download files.
for urld in urls2dwn:
    # Extract the filename from the URL.
    ntw = urld.split('/')[-1]

    # Construct the complete path to save the file in the local directory.
    file_path = os.path.join(local_directory, ntw)

    # Send an HTTP GET request to the URL.
    resp = requests.get(urld)

    # Check if the response is successful (status code 200).
    if resp.status_code == 200:
        # Write the content to the file in binary mode.
        with open(file_path, "wb") as file:
            file.write(resp.content)
        print(f"File '{ntw}' downloaded and saved to '{local_directory}'.")
    else:
        print(f"Failed to download '{ntw}' from the URL: {urld}")
```

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
from satpy.scene import Scene
from satpy import find_files_and_readers
```

Searching for GOES-R L1B Data

In [ ]:
```
# Corrected path with double backslashes or raw string
base_dir = "Output_data/ABI-L1b-RadF/s20210992350171"

# Use the corrected path and reader to find files and readers
fGRl1b = find_files_and_readers(base_dir=base_dir, reader='abi_l1b')

# List the found files and associated readers
fGRl1b
```

Searching for GOES-R L2 CMIPC Data

In [ ]:
```
urls2dwn = ['https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L
  'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
  'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
```

```
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
 'https://noaa-goes16.s3.amazonaws.com/ABI-L2-CMIPC/2018/047/19/OR_ABI-L2-CMIPC-M3
```

In [ ]:

```
import requests
import os

# Specify the local directory where you want to save the files.
# local_directory = input("Enter the path to the download folder: ")
local_directory = "Output_data/ABI-L2-CMIPC/s20180471917"

# Ensure that the local directory exists; create it if it doesn't.
os.makedirs(local_directory, exist_ok=True)

# Iterate through the URLs and download files.
for urld in urls2dwn:
    # Extract the filename from the URL.
    ntw = urld.split('/')[-1]

    # Construct the complete path to save the file in the local directory.
    file_path = os.path.join(local_directory, ntw)

    # Send an HTTP GET request to the URL.
    resp = requests.get(urld)

    # Check if the response is successful (status code 200).
    if resp.status_code == 200:
        # Write the content to the file in binary mode.
        with open(file_path, "wb") as file:
            file.write(resp.content)
        print(f"File '{ntw}' downloaded and saved to '{local_directory}'.")
    else:
        print(f"Failed to download '{ntw}' from the URL: {urld}")
```

In [ ]:

```
# Define the path to the FCI test data folder

base_dir = "Output_data/ABI-L2-CMIPC/s20180471917"

# Use the defined path and reader ('abi_l2_nc') to find files and associated reader
fGRl2 = find_files_and_readers(base_dir=base_dir, reader='abi_l2_nc')

# List the found files and associated readers
fGRl2
```

Searching for Native MSG Data

In [ ]:
```
from datetime import datetime
# Create a datetime object representing October 28, 2023, 3:30 PM
date_time = datetime(2023, 10, 28, 15, 30)
```

In [ ]:
```
# Import the 'glob' function from the 'glob' module, which is used for file and dir
from glob import glob
```

> **Attention:**
>
> **SatPy** always expects the original file names!
>
> So, do not change them when saving the data on your local machine.
> Otherwise, SatPy will not be able to open the files.

In [ ]:
```
# Create a Scene object 'scn' using a list of filenames 'fGRl1b'
scn = Scene(filenames=fGRl1b)
```

In [ ]:
```
# Retrieve and display the attributes of the 'scn' Scene object
scn.attrs
```

Below code shows differentbands avaialble in the image file corresponding to different wavelengths

In [ ]:
```
# Retrieve and list all the dataset names available within the 'scn' Scene object
scn.all_dataset_names()
```

In [ ]:
```
# Retrieve and list the keys (dataset names) available within the 'scn' Scene objec
# These keys correspond to different bands or datasets. For more details about thes
# http://www.geo-web.org.uk/sats.php
scn.keys()
```

In [ ]:
```
# Load the dataset named "C13" into the 'scn' Scene object.
scn.load(["C13"])
```

In [ ]:
```
# Display the data from the "C13" dataset within the 'scn' Scene object.
scn.show("C13")
```

In [ ]:
```
# Enable inline plotting using Matplotlib for the Jupyter Notebook (%matplotlib inl
%matplotlib inline

# Plot and display the "C13" dataset/band from the 'scn' Scene object as an image.
scn["C13"].plot.imshow()
```

In [ ]:
```
# Import the Matplotlib library as 'plt' for creating plots and visualizations
import matplotlib.pyplot as plt
```

In [ ]:
```
# Create a Matplotlib figure and axis with a specified size (10x10 inches)
fig, ax = plt.subplots(figsize=(10, 10))

# Display the data from the "C13" dataset in grayscale ("Greys_r" colormap)
plt.imshow(scn["C13"].values, cmap="Greys_r")

# Turn off the axis labels and ticks
ax.set_axis_off()

# Add a colorbar to the plot with a specified fraction of the original size
plt.colorbar(fraction=0.04)

# Show the plot
plt.show()
```

In [ ]:
```
# Access and retrieve the data from the "C13" dataset within the 'scn' Scene object
scn["C13"]
```

In [ ]:
```
# Load the band at 10.3µm as radiances (in mW m-2 sr-1(cm-1)-1):
scn.load([10.3], calibration=["radiance"])

# Load the band at 10.3µm as brightness temperatures (in Kelvin, K):
scn.load([10.3], calibration=["brightness_temperature"])
```

In [ ]:
```
# Retrieve and list the keys (dataset names) available within the 'scn' Scene objec
scn.keys()
```

In [ ]:
```
# Retrieve the wavelength and calibration information for the second dataset (index
# The [1] index refers to the second dataset, and ['wavelength'] and ['calibration'
scn.keys()[1]['wavelength'][1], scn.keys()[1]['calibration']
```

In [ ]:
```
# Create a Matplotlib figure with two subplots side by side (1 row, 2 columns), spe
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# Display the data from the second dataset (index 1) as an image in the first subpl
im1 = ax1.imshow(scn[scn.keys()[1]].values, cmap="Greys_r")
ax1.set_axis_off()  # Turn off axis labels and ticks for the first subplot

# Display the data from the first dataset (index 0) as an image in the second subpl
im2 = ax2.imshow(scn[scn.keys()[0]].values, cmap="Greys_r")
ax2.set_axis_off()  # Turn off axis labels and ticks for the second subplot

# Set titles for the subplots to indicate the type of data displayed.
ax2.set_title("10.3µm Brightness temperature")
ax1.set_title("10.3µm Radiance")

# Add colorbars to the subplots with a specified fraction of the original size.
fig.colorbar(im1, ax=ax1, fraction=0.05)
fig.colorbar(im2, ax=ax2, fraction=0.05)

# Show the figure with the two subplots.
plt.show()
```

Question: How can you plot or access the data for the 'C13' dataset?"

In [ ]:
```
# Solution:
# To plot the 'C13' dataset:
scn['C13'].plot.imshow()

# To access the data values for the 'C13' dataset:
c13_data = scn['C13'].values
```

In [ ]:
```
# Retrieve the 'x' coordinate information for the 'C13' dataset.
x_coordinates = scn["C13"].x
```

In [ ]:
```
# define palette (matplotlib style)
cmap = ['#ffffff', '#ffffff', '#ffffff', '#ffffff', '#ffffff', '#b6ffb6', '#79ff79'
        '#0028a2', '#000079', '#fbfb00', '#e7e700', '#d2d200', '#baba00', '#a6a600'
        '#aaaaaa', '#a6a6a6', '#9e9e9e', '#969696', '#8e8e8e', '#868686', '#7d7d7d'
        '#313131', '#282828', '#202020', '#181818', '#141414', '#000000', '#000000'
```

In [ ]:
```
# Retrieve the area definition (spatial information) associated with the 'C13' data
area_def = scn['C13'].attrs['area']
```

In [ ]:
```
# Convert the area definition to a Cartopy Coordinate Reference System (CRS).
cartopy_crs = area_def.to_cartopy_crs()
```

In [ ]:
```
# Display the results of the 'C13' dataset on a map using Cartopy and Matplotlib.
# Convert the area definition to a Cartopy CRS for proper georeferencing.
crs = area_def.to_cartopy_crs()

# Create a Matplotlib figure with a specified size.
fig = plt.figure(figsize=(10, 10))

# Create a map axis using Cartopy with coastlines and gridlines.
ax = plt.axes(projection=crs)
ax.coastlines()
ax.gridlines()

# Display the 'C13' dataset on the map with the correct georeferencing.
plt.imshow(scn['C13'], transform=crs, extent=crs.bounds, origin='upper')

# Add a colorbar to the plot with units information from the dataset's attributes.
plt.colorbar(label=scn['C13'].attrs['units'])

# Show the map with the dataset.
plt.show()
```

In [ ]:
```
import numpy as np
```

In [ ]:
```
# Define custom color levels and colormap for visualization.
# This code sets up custom levels, a normalization method, and a colormap for the v
levels = np.linspace(-109, 56, num=len(cmap))  # Define custom levels for color map
norm = plt.cm.colors.BoundaryNorm(levels, len(levels))  # Normalize data based on l
irmap = plt.cm.colors.ListedColormap(cmap)  # Create a custom colormap.

# Display the results of the 'C13' dataset on a map using Cartopy and Matplotlib.
# Convert the area definition to a Cartopy CRS for proper georeferencing.
```

```
crs = area_def.to_cartopy_crs()

# Create a Matplotlib figure with a specified size.
fig = plt.figure(figsize=(10, 10))

# Create a map axis using Cartopy with coastlines and gridlines.
ax = plt.axes(projection=crs)
ax.coastlines()
ax.gridlines()

# Display the 'C13' dataset on the map with the correct georeferencing and custom c
# Set the minimum and maximum values for color mapping using 'vmin' and 'vmax'.
plt.imshow(scn['C13'], transform=crs, extent=crs.bounds, origin='upper',
           vmin=-109 + 273.15, vmax=56 + 273.15, cmap=irmap)

# Add a colorbar to the plot with units information from the dataset's attributes.
plt.colorbar(label=scn['C13'].attrs['units'])

# Show the map with the dataset.
plt.show()
```

Cropping

In [ ]:
```
# Crop the scene to a specific region defined by latitude and longitude values.
# The bounding box is defined as (lon_min, lat_min, lon_max, lat_max).
# In this case, the region spans from longitude -93.5° to -76.0° and from latitude

# Crop the scene to the specified bounding box.
scn_cropped = scn.crop(ll_bbox=(-93.5, 5.5, -76.0, 19.5))

# Display the cropped dataset for the 'C13' channel.
scn_cropped.show("C13")
```

In [ ]:
```
# Retrieve the area definition (spatial information) associated with the 'C13' data
area_def_c = scn_cropped['C13'].attrs['area']
```

In [ ]:
```
# Display the results of the 'C13' dataset within the cropped region using Cartopy
# Convert the area definition of the cropped scene to a Cartopy CRS for proper geor
crs = area_def_c.to_cartopy_crs()

# Create a Matplotlib figure with a specified size.
fig = plt.figure(figsize=(10, 10))

# Create a map axis using Cartopy with coastlines and gridlines.
ax = plt.axes(projection=crs)
ax.coastlines()
ax.gridlines()

# Display the 'C13' dataset within the cropped region on the map.
# Set the minimum and maximum values for color mapping using 'vmin' and 'vmax'.
# Apply the custom colormap 'irmap' defined earlier.
plt.imshow(scn_cropped['C13'], transform=crs, extent=crs.bounds, origin='upper',
           vmin=-109 + 273.15, vmax=56 + 273.15, cmap=irmap)
```

```
# Add a colorbar to the plot with units information from the dataset's attributes.
plt.colorbar(label=scn_cropped['C13'].attrs['units'])

# Show the map with the 'C13' dataset within the cropped region.
plt.show()
```