# Homework 4 Task 2 - False Color Imagery Analysis

**Author**: Jay Kim

**Github Repository**: https://github.com/jwonyk/eds220-hwk4

## About

### Purpose

The purpose of this notebook is to explore and analyze wildfire impacts in the Palisade-Eaton region using a fire perimeter data and Landsat 8 remote sensing imagery. The analysis includes importing, examining combination of datasets, restoring missing CRS information, generate both true and false color images, and plot the map to visualize. After reviewing this notebook, the audience will understand how to manipulate a geospatial raster and vector data, interpret spectral bands, and create visualizations the current environmental analysis in Python.

### Highlights

- Restore missing CRS from a Landsat NetCDF file
- Create true and false-color images to visualize vegetation, burned areas, and the landscape condition.
- Identify cloud outliers and handle NaN values to improve image clarity
- Overlay false-color imagery with the fire perimeter to illustrate aftermath of wildfire

### About the data

#### Fire Perimeter Dataset

The fire perimeter dataset contains dissolved burn boundaries for the Eaton and Palisades fires. The original data comes from NIFC FIRIS (Fire Integrated Real-time Intelligence System). The data provides daily fire growth during the event of wildfire. The dataset is accessed through Los Angeles GeoHub and it is also accesscible from NIFC FIRIS public ArcGIS service.

#### Landsat NetCDF Dataset

The Landsat used in this analysis is a Landsat 8 Collection Level-2 Surface Reflectance product accessed through the Microsoft Planetary Computer Data Catalogue. The dataset includes multiple spectral bands stored in a NetCDF structure, CRS information,

coordinate dimensions, and metadata. The spectral bands will assist to create true and false-color composite images for post-fire assessment.

### Reference


- Los Angeles GeoHub / NIFC FIRIS. (2025). *Palisades–Eaton dissolved fire perimeters* [data file]. Available: https://geohub.lacity.org/maps/ad51845ea5fb4eb483bc2a7c38b2370c/about. [Accessed: Nov. 15, 2025]

- U.S. Geological Survey. *Landsat Collection 2 Level-2 Surface Reflectance (Microsoft Planetary Computer version)* [data file]. Available: https://planetarycomputer.microsoft.com/dataset/landsat-c2-l2. [Accessed: Nov. 15, 2025]


In [ ]:
```
# Import packages for this project
import os
import numpy as np
import pandas as pd
import geopandas as gpd
import xarray as xr
import rioxarray
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

# Set pandas to display all columns
pd.set_option("display.max_columns", None)
```

## Fire Perimeter Data Exploration

### Preview

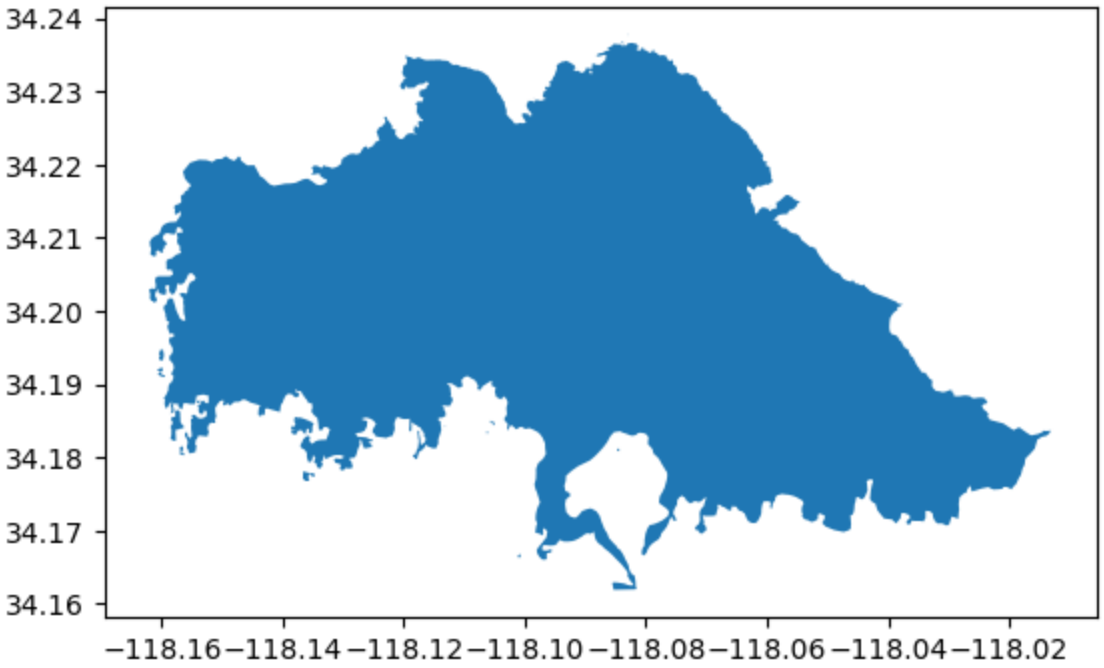
In this section, I will load and examine the Eaton and Palisades fire perimeter datasets. I will check their CRS, inspect the attribute table, and visualize the geometries to understanding the CRS and structure is essential before overlaying them with raster data.

In [ ]:
```
# Load fire perimeter datasets
eaton = gpd.read_file(os.path.join('data',
                                   'fire_perimeters',
                                   'Eaton_Perimeter_20250121.geojson'))

palisades = gpd.read_file(os.path.join('data',
                                   'fire_perimeters',
                                   'Palisades_Perimeter_20250121.geojson'
```

In [ ]:
```
# Explore Eaton datasets
eaton.info(), eaton.crs, eaton.geometry, eaton.plot()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   OBJECTID       20 non-null     int64
 1   type           20 non-null     object
 2   Shape__Area    20 non-null     float64
 3   Shape__Length  20 non-null     float64
 4   geometry       20 non-null     geometry
dtypes: float64(2), geometry(1), int64(1), object(1)
memory usage: 932.0+ bytes
```

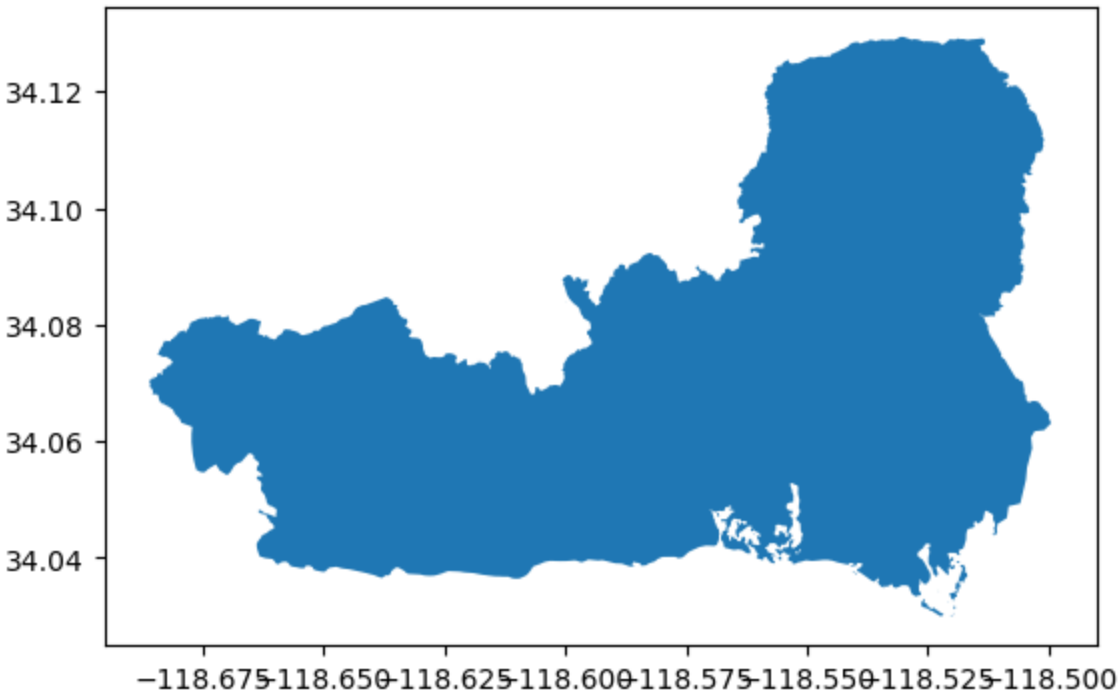
Out[ ]:
```
(None,
 <Geographic 2D CRS: EPSG:4326>
 Name: WGS 84
 Axis Info [ellipsoidal]:
 - Lat[north]: Geodetic latitude (degree)
 - Lon[east]: Geodetic longitude (degree)
 Area of Use:
 - name: World.
 - bounds: (-180.0, -90.0, 180.0, 90.0)
 Datum: World Geodetic System 1984 ensemble
 - Ellipsoid: WGS 84
 - Prime Meridian: Greenwich,
 0     POLYGON ((-118.10094 34.16681, -118.10090 34.1...
 1     POLYGON ((-118.13596 34.17789, -118.13593 34.1...
 2     POLYGON ((-118.15626 34.18045, -118.15643 34.1...
 3     POLYGON ((-118.08442 34.18090, -118.08445 34.1...
 4     POLYGON ((-118.15659 34.18148, -118.15659 34.1...
 5     POLYGON ((-118.10616 34.18327, -118.10622 34.1...
 6     POLYGON ((-118.11317 34.18429, -118.11317 34.1...
 7     POLYGON ((-118.10578 34.18497, -118.10576 34.1...
 8     POLYGON ((-118.13770 34.18523, -118.13766 34.1...
 9     POLYGON ((-118.15697 34.18552, -118.15696 34.1...
 10    POLYGON ((-118.15394 34.18781, -118.15378 34.1...
 11    POLYGON ((-118.16031 34.19172, -118.16031 34.1...
 12    POLYGON ((-118.15995 34.19181, -118.16006 34.1...
 13    POLYGON ((-118.15984 34.19488, -118.15984 34.1...
 14    POLYGON ((-118.05930 34.21416, -118.05942 34.2...
 15    POLYGON ((-118.09846 34.22899, -118.09844 34.2...
 16    POLYGON ((-118.09855 34.22912, -118.09859 34.2...
 17    POLYGON ((-118.08140 34.23629, -118.08129 34.2...
 18    POLYGON ((-118.08369 34.23651, -118.08376 34.2...
 19    POLYGON ((-118.08303 34.23783, -118.08303 34.2...
 Name: geometry, dtype: geometry,
 <Axes: >)
```




In [ ]:
```
# Explore Palisades datasets
palisades.info(), palisades.crs, palisades.geometry, palisades.plot()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   OBJECTID       21 non-null     int64
 1   type           21 non-null     object
 2   Shape__Area    21 non-null     float64
 3   Shape__Length  21 non-null     float64
 4   geometry       21 non-null     geometry
dtypes: float64(2), geometry(1), int64(1), object(1)
memory usage: 972.0+ bytes
```

Out[ ]:
```
(None,
 <Geographic 2D CRS: EPSG:4326>
 Name: WGS 84
 Axis Info [ellipsoidal]:
 - Lat[north]: Geodetic latitude (degree)
 - Lon[east]: Geodetic longitude (degree)
 Area of Use:
 - name: World.
 - bounds: (-180.0, -90.0, 180.0, 90.0)
 Datum: World Geodetic System 1984 ensemble
 - Ellipsoid: WGS 84
 - Prime Meridian: Greenwich,
 0     POLYGON ((-118.51962 34.03061, -118.51962 34.0...
 1     POLYGON ((-118.51944 34.03176, -118.51944 34.0...
 2     POLYGON ((-118.52011 34.03244, -118.52011 34.0...
 3     POLYGON ((-118.52061 34.03235, -118.52063 34.0...
 4     POLYGON ((-118.52560 34.03302, -118.52560 34.0...
 5     POLYGON ((-118.52286 34.03427, -118.52283 34.0...
 6     POLYGON ((-118.52276 34.03435, -118.52273 34.0...
 7     POLYGON ((-118.51818 34.03739, -118.51820 34.0...
 8     POLYGON ((-118.53964 34.03715, -118.53971 34.0...
 9     POLYGON ((-118.51748 34.03607, -118.51752 34.0...
 10    POLYGON ((-118.51841 34.03892, -118.51841 34.0...
 11    POLYGON ((-118.51960 34.03911, -118.51965 34.0...
 12    POLYGON ((-118.51970 34.03923, -118.51970 34.0...
 13    POLYGON ((-118.51874 34.03947, -118.51869 34.0...
 14    POLYGON ((-118.55644 34.04015, -118.55655 34.0...
 15    POLYGON ((-118.55547 34.04031, -118.55531 34.0...
 16    POLYGON ((-118.51155 34.04422, -118.51149 34.0...
 17    POLYGON ((-118.56093 34.04408, -118.56097 34.0...
 18    POLYGON ((-118.56856 34.04770, -118.56821 34.0...
 19    POLYGON ((-118.60003 34.08403, -118.60003 34.0...
 20    POLYGON ((-118.52862 34.12894, -118.52837 34.1...
 Name: geometry, dtype: geometry,
 <Axes: >)
```




### Fire Perimeter Data Summary

- Both Eaton and Palisades have a matching CRS (EPSG: 4326)
- Both Eaton and Palisades have **geophraphic** CRS.
- Both datasets have same structure:
  - GIS fields (OBJECTID, type, Shape_Area, Shape_Length)
  - Polygon geometry representing each fire final perimeter.

## NetCDF Data Import & Exploration

### Preview

I will load the Landsat 8 NetCDF file using `xr.open_dataset()` and examined its dimensions, coordinates, and spectral bands. The goal is to understand what information is included before restoring the missing CRS.

In [ ]:
```
# Import NetCDF dataset
landsat = xr.open_dataset("data/landsat8-2025-02-23-palisades-eaton.nc")

# Inspect the dataset
landsat
```

Out[ ]: xarray.Dataset

▶ Dimensions: (**y**: 1418, **x**: 2742)

▼ Coordinates:

| | | | |
|---|---|---|---|
| **y** | (y) | float64 | 3.799e+06 3.799e+06 ... 3.757e... |
| **x** | (x) | float64 | 3.344e+05 3.344e+05 ... 4.166e... |
| time | () | datetime64[ns] | ... |

▼ Data variables:

| | | | |
|---|---|---|---|
| red | (y, x) | float32 | ... |
| green | (y, x) | float32 | ... |
| blue | (y, x) | float32 | ... |
| nir08 | (y, x) | float32 | ... |
| swir22 | (y, x) | float32 | ... |
| spatial_ref | () | int64 | ... |

▶ Indexes: (2)

▶ Attributes: (0)

In [ ]:
```
# Explore the landsat data
landsat.dims, landsat.coords, landsat.data_vars, landsat.info()
```

```
xarray.Dataset {
dimensions:
        y = 1418 ;
        x = 2742 ;

variables:
        float64 y(y) ;
                y:units = metre ;
                y:resolution = -30.0 ;
                y:crs = EPSG:32611 ;
                y:axis = Y ;
                y:long_name = y coordinate of projection ;
                y:standard_name = projection_y_coordinate ;
        float64 x(x) ;
                x:units = metre ;
                x:resolution = 30.0 ;
                x:crs = EPSG:32611 ;
                x:axis = X ;
                x:long_name = x coordinate of projection ;
                x:standard_name = projection_x_coordinate ;
        datetime64[ns] time() ;
        float32 red(y, x) ;
                red:grid_mapping = spatial_ref ;
        float32 green(y, x) ;
                green:grid_mapping = spatial_ref ;
        float32 blue(y, x) ;
                blue:grid_mapping = spatial_ref ;
        float32 nir08(y, x) ;
                nir08:grid_mapping = spatial_ref ;
        float32 swir22(y, x) ;
                swir22:grid_mapping = spatial_ref ;
        int64 spatial_ref() ;
                spatial_ref:crs_wkt = PROJCS["WGS 84 / UTM zone 11N",GEOGCS
["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY
["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY["E
PSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUT
HORITY["EPSG","4326"]],PROJECTION["Transverse_Mercator"],PARAMETER["latitude
_of_origin",0],PARAMETER["central_meridian",-117],PARAMETER["scale_factor",
0.9996],PARAMETER["false_easting",500000],PARAMETER["false_northing",0],UNIT
["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northing",NO
RTH],AUTHORITY["EPSG","32611"]] ;
                spatial_ref:semi_major_axis = 6378137.0 ;
                spatial_ref:semi_minor_axis = 6356752.314245179 ;
                spatial_ref:inverse_flattening = 298.257223563 ;
                spatial_ref:reference_ellipsoid_name = WGS 84 ;
                spatial_ref:longitude_of_prime_meridian = 0.0 ;
                spatial_ref:prime_meridian_name = Greenwich ;
                spatial_ref:geographic_crs_name = WGS 84 ;
                spatial_ref:horizontal_datum_name = World Geodetic System 19
84 ;
                spatial_ref:projected_crs_name = WGS 84 / UTM zone 11N ;
                spatial_ref:grid_mapping_name = transverse_mercator ;
                spatial_ref:latitude_of_projection_origin = 0.0 ;
                spatial_ref:longitude_of_central_meridian = -117.0 ;
                spatial_ref:false_easting = 500000.0 ;
                spatial_ref:false_northing = 0.0 ;
```

```
                spatial_ref:scale_factor_at_central_meridian = 0.9996 ;
                spatial_ref:spatial_ref = PROJCS["WGS 84 / UTM zone 11N",GEO
GCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHOR
ITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY
["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],
AUTHORITY["EPSG","4326"]],PROJECTION["Transverse_Mercator"],PARAMETER["latit
ude_of_origin",0],PARAMETER["central_meridian",-117],PARAMETER["scale_facto
r",0.9996],PARAMETER["false_easting",500000],PARAMETER["false_northing",0],U
NIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northin
g",NORTH],AUTHORITY["EPSG","32611"]] ;
                spatial_ref:GeoTransform = 334395.0 30.0 0.0 3799065.0 0.0 -
30.0 ;

// global attributes:
}
```

Out[ ]:
```
(FrozenMappingWarningOnValuesAccess({'y': 1418, 'x': 2742}),
 Coordinates:
   * y        (y) float64 11kB 3.799e+06 3.799e+06 ... 3.757e+06 3.757e+06
   * x        (x) float64 22kB 3.344e+05 3.344e+05 ... 4.166e+05 4.166e+05
     time     datetime64[ns] 8B ...,
 Data variables:
     red          (y, x) float32 16MB ...
     green        (y, x) float32 16MB ...
     blue         (y, x) float32 16MB ...
     nir08        (y, x) float32 16MB ...
     swir22       (y, x) float32 16MB ...
     spatial_ref  int64 8B ...,
 None)
```

## Landsat NetCDF Dataset Summary

The description in the metadata is explain in the following:

- Dimension:
  - `x` : 2742 pixels wide (east to west) and has 30 m resolution
  - `y` : 1418 pixels tall (north to south) and has -30 m resolution
  - `time` : one value in datetime64 format
- Coordinates:
  - `x` : Projected coordinates in meters and includes resolution, axis label, and CRS
  - `y` : Projected coordinates in meters and includes resolution, axis label, and CRS
  - `time` : Acquisition date / time of the Landsat scene
  - `spatial_ref` : An integer placeholder coordinate with projection metadata such as CRS WKT, ellipsoid data, transform, latitude and longtitude, etc.
- Data Variables:
  - All colors have reflectance values mapped through the `spacial_ref`
  - `red` : float32 format; red band
  - `green` : float32 format; green band
  - `blue` : float32 format; blue band

    - `nir08` : float32 format; Near-Infrared (NIR) band for vegetation monitoring and false-color imagery
    - `swir22` : float32 format; Shortwave Infrared 2 (SWIR2) for burn areas and moisture analysis
    - `spatial_ref` : int64 format; variable contains projection metadata
- Additional note:
    - All CRS information is stored inside the `spatial_ref` variable
    - CRS set to WGS 84 (EPSG: 32611) and it is **projected** coordinate system (Area of use: Between 120°W and 114°W, northern hemisphere between equator and 84°N, onshore and offshore. Canada - Alberta; British Columbia (BC); Northwest Territories (NWT); Nunavut. Mexico. United States (USA))

# Restore Geospatial Information (CRS Recovery)

### Preview

The NetCDF file contains projection information inside the `spatial_ref` variable. `rio.crs` initially shows that no CRS is assigned. I will extract the WKT projection from `spatial_ref.crs_wkt` and write it back to the dataset using `rio.write_crs()`.

### a. Check if dataset has a CRS

In [ ]:
```
# Use `rio.crs` to print what is the CRS of this dataset
landsat.rio.crs # Does not return anything
```

### b. Retrieve CRS from `spatial_ref`

In [ ]:
```
# Print the CRS by using accesing the `spatial_ref.crs_wkt`
landsat.spatial_ref.crs_wkt # Projected - WGS 84
```

Out[ ]:
```
'PROJCS["WGS 84 / UTM zone 11N",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID
["WGS 84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPS
G","6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.
0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]],PROJEC
TION["Transverse_Mercator"],PARAMETER["latitude_of_origin",0],PARAMETER["ce
ntral_meridian",-117],PARAMETER["scale_factor",0.9996],PARAMETER["false_eas
ting",500000],PARAMETER["false_northing",0],UNIT["metre",1,AUTHORITY["EPS
G","9001"]],AXIS["Easting",EAST],AXIS["Northing",NORTH],AUTHORITY["EPSG","3
2611"]]'
```

### c. Write the CRS back into the dataset

In [ ]:
```
# Use the `.rio.write_crs()` to write CRS in `rioxarray`
landsat = landsat.rio.write_crs(landsat.spatial_ref.crs_wkt)
```

### d. Print the CRS for the updated dataset

In [ ]:
```
# Check the CRS
landsat.rio.crs
```

Out[ ]:
```
CRS.from_wkt('PROJCS["WGS 84 / UTM zone 11N",GEOGCS["WGS 84",DATUM["WGS_198
4",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHOR
ITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["de
gree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","432
6"]],PROJECTION["Transverse_Mercator"],PARAMETER["latitude_of_origin",0],PA
RAMETER["central_meridian",-117],PARAMETER["scale_factor",0.9996],PARAMETER
["false_easting",500000],PARAMETER["false_northing",0],UNIT["metre",1,AUTHO
RITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northing",NORTH],AUTHORITY
["EPSG","32611"]]')
```

### CRS Recovery Summary

- The Landsat NetCDF file did not had CRS assigned through the `.rio` accessor.
  - `landsat.rio.crs` did not return output
  - The Landsat dataset has `spatial_ref` variable that contains complete projection information in `crs_wkt` attribute
  - The metadata contains the CRS information as **projected** WGS 84 / UTM Zone 11N (EPSG: 325611)
- The following process has been used to restore the geospatial information:
  - Extract WKT string from `spatial_ref.crs_wkt`
  - Apply dataset using `rio.write__crs()`
  - After writing the CRS, we confirm `landsat.rio.crs` returning the expected projection and recognizing as a geospatial raster

## True Color Image

### Preview

I will select the red, green, and blue bands and converted them into a 3-band array. The initial plot produces warnings due to bright cloud outliers, so I will use the `robust=True` parameter to ignore extreme values. At the end, I will identify and replace `NaN` values to create a clean final RGB image.

### a. Select red, green, blue bands without creating new variable

In [ ]:
```
# Select the red, green, and blue variables
rgb_data = landsat[['red', 'green', 'blue']] # Use double bracket for list

# Convert it using the `to_array()`
rgb_array = rgb_data.to_array()

# Use `.plot.imshow()` to create an RGB image with the data
image = rgb_array.plot.imshow()

# Get the Axes object from the image
```

```
ax = image.axes

# Remove ticks
ax.set_xticks([])
ax.set_yticks([])

# Replace x and y label
ax.set_xlabel("Easting (meters)", fontsize = 12)
ax.set_ylabel("Northing (meters)", fontsize = 12)

# Add title
ax.set_title("True Color Image from Landsat 8 (Feb 23, 2025)",
             fontsize = 14)

# Improve layout so elements do not overlap
plt.tight_layout()
```

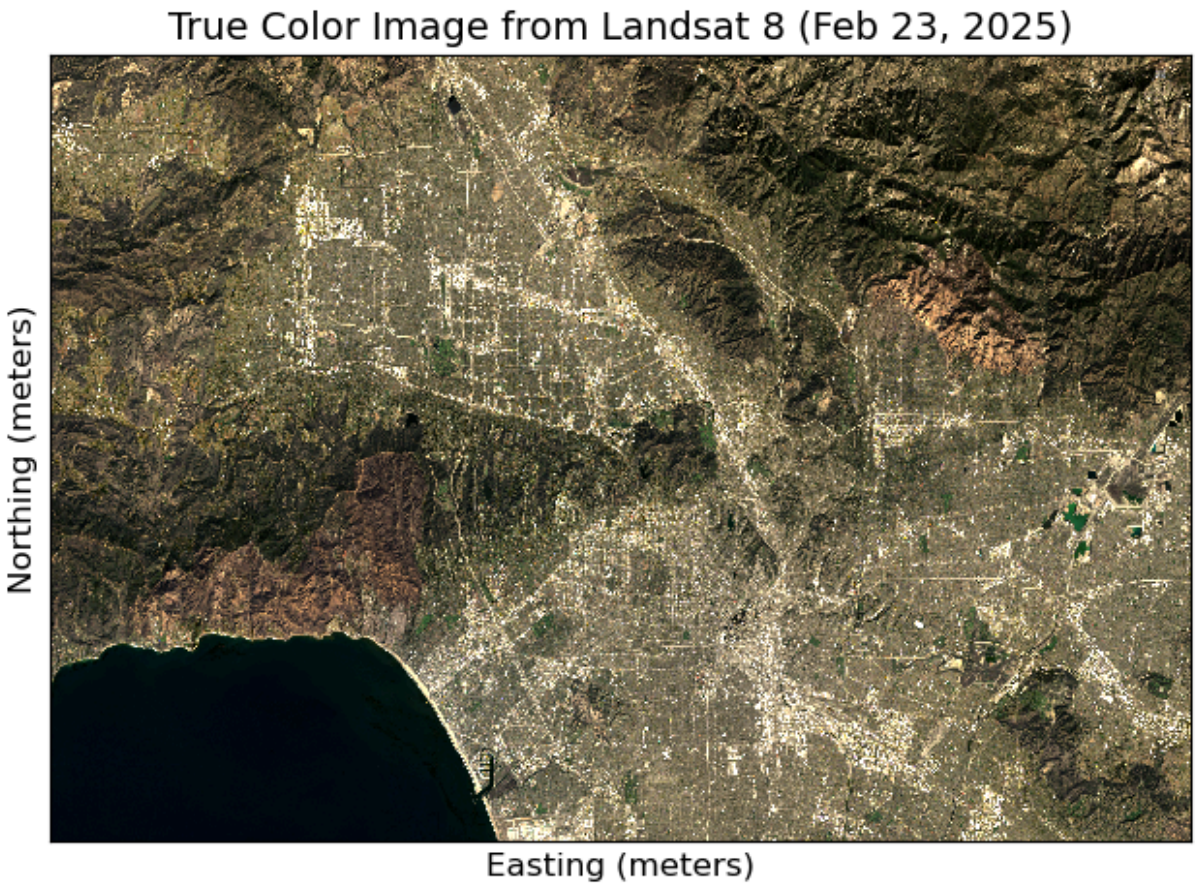
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
/opt/anaconda3/envs/eds220-env/lib/python3.11/site-packages/matplotlib/cm.p
y:478: RuntimeWarning: invalid value encountered in cast
  xx = (xx * 255).astype(np.uint8)
```

### b. Adjust the scale used for plotting the bands

- Use `robust = True` parameter inside the `.imshow()`
- The reason the prior plot had empty plot due to the fact that there is outlier. Cloud outliers caused the other values to be squished when plotting

In [ ]:
```
# Add `robust = True` to the parameter of `.plot.imshow()`
# This parameter will ignore extreme pixel outliers
image = rgb_array.plot.imshow(robust = True)

# Get the Axes object from the image
ax = image.axes

# Remove both axis ticks
ax.set_xticks([])
ax.set_yticks([])

# Replace x and y label
ax.set_xlabel("Easting (meters)", fontsize = 12)
ax.set_ylabel("Northing (meters)", fontsize = 12)

# Add title
ax.set_title("True Color Image from Landsat 8 (Feb 23, 2025)",
             fontsize = 14)

# Improve layout so elements do not overlap
plt.tight_layout()
```

```
/opt/anaconda3/envs/eds220-env/lib/python3.11/site-packages/matplotlib/cm.p
y:478: RuntimeWarning: invalid value encountered in cast
  xx = (xx * 255).astype(np.uint8)
```




### c. Identify NaNs

In [ ]:
```
# Identify which bands have `NaN` values
for band in ['red', 'green', 'blue', 'nir08', 'swir22']:
    check = np.isnan(landsat[band]).any().item() # Convert to boolean
    print(band, check)
```

```
red False
green True
blue True
nir08 False
swir22 False
```

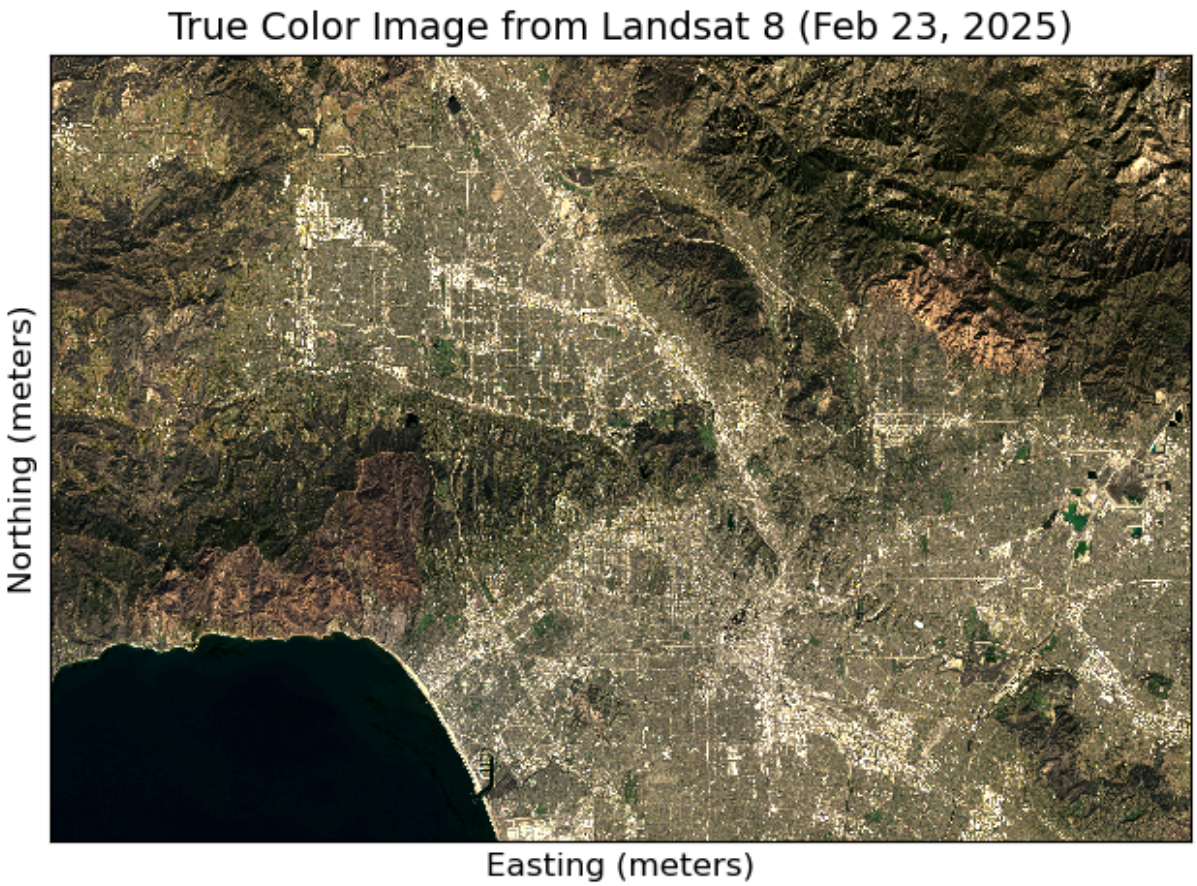
### d. Fill `NaN` values with 0

In [ ]:
```
# Clean the dataset by filling `NaN` with 0
landsat_clean = landsat.fillna(0)
```

### e. Create a new true color image that gets plotted without warnings

In [ ]:
```
# Reassign with filled dataset
rgb_clean = landsat_clean[['red', 'green', 'blue']]

# Return only the plot by adding semicolon
image = rgb_clean.to_array().plot.imshow(robust = True);
```

```
# Get the Axes object from the image
ax = image.axes

# Remove both axis ticks
ax.set_xticks([])
ax.set_yticks([])

# Replace x and y label
ax.set_xlabel("Easting (meters)", fontsize = 12)
ax.set_ylabel("Northing (meters)", fontsize = 12)

# Add title
ax.set_title("True Color Image from Landsat 8 (Feb 23, 2025)",
             fontsize = 14)

# Improve layout so elements do not overlap
plt.tight_layout()
```




### f. True Color Image Comparison

- Plot from (a) had the initial true color produced warnings and did not show anything in the plot itself due to the fact that there are `NaN` values and bright cloud outliers affecting the scaling.
  - Use `robust = True` to surpress the influence of extreme pixel values
  - Identify `NaN` and replaced with zeros using `.fillna()`, the image render clean without any warnings

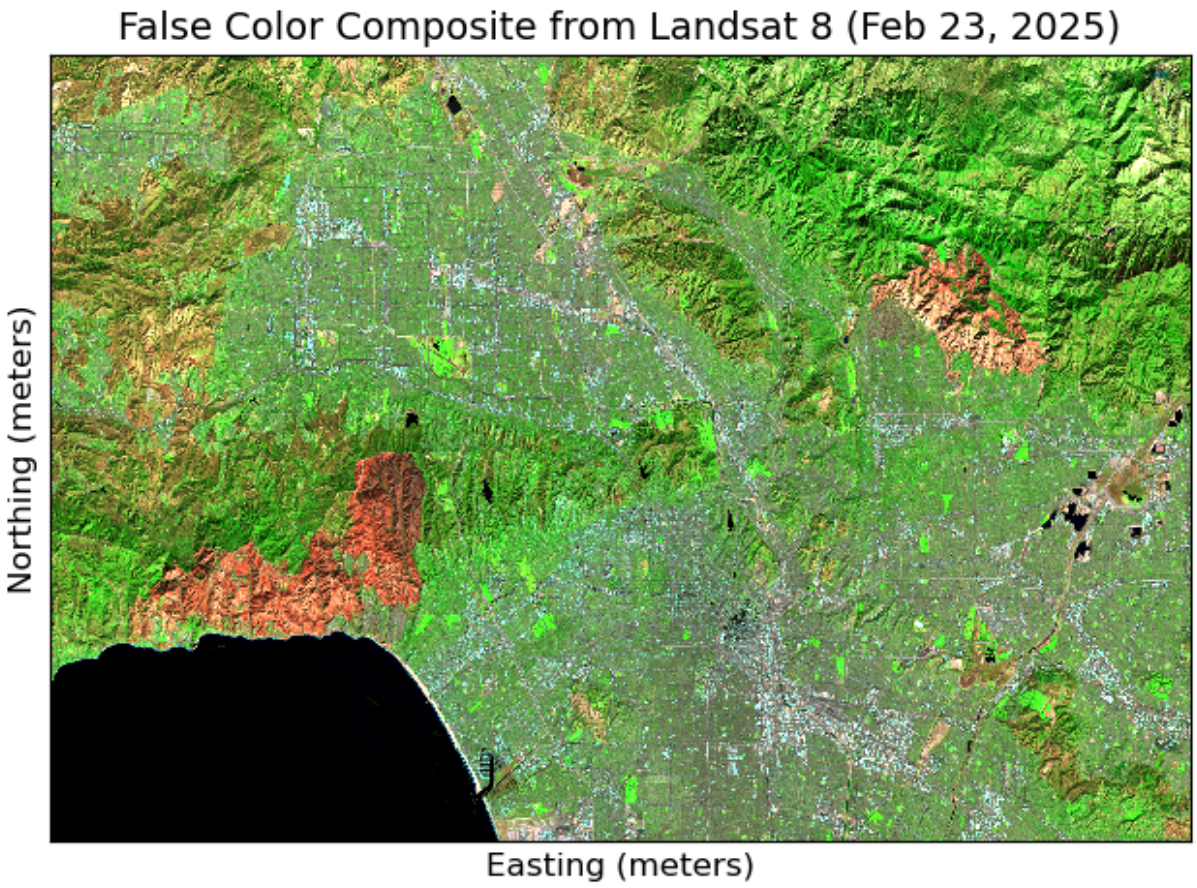
- Add semicolon to return only the plot

# False Color Image

## Preview

False color composites help highlight vegetation health and burn severity. I will plot a `SWIR2`, `NIR`, `Red` composite without creating new variables.

In [ ]:
```
# Plot false color composite without creating new variable
# `image` variable was used in prior plot & return only the plot
image = (landsat_clean[['swir22', 'nir08', 'red']]
                                                    .to_array()
                                                    .plot.imshow(robust = True

# Get the Axes object from the image
ax = image.axes

# Remove both axis ticks
ax.set_xticks([])
ax.set_yticks([])

# Replace x and y label
ax.set_xlabel("Easting (meters)", fontsize = 12)
ax.set_ylabel("Northing (meters)", fontsize = 12)

# Add title
ax.set_title("False Color Composite from Landsat 8 (Feb 23, 2025)",
             fontsize = 14)

# Improve layout so elements do not overlap
plt.tight_layout()
```




### False Color Summary

- False color composite is created with following:
  - SWIR2, NIR, and Red bands
  - This enhances the burned area and fire impacts
  - There are signs of vegetation which was not as affected by fire
  - The result provides clearer differentiation of land cover and make it easier to assess burn severity.

## Map with Fire Perimeters

### Preview

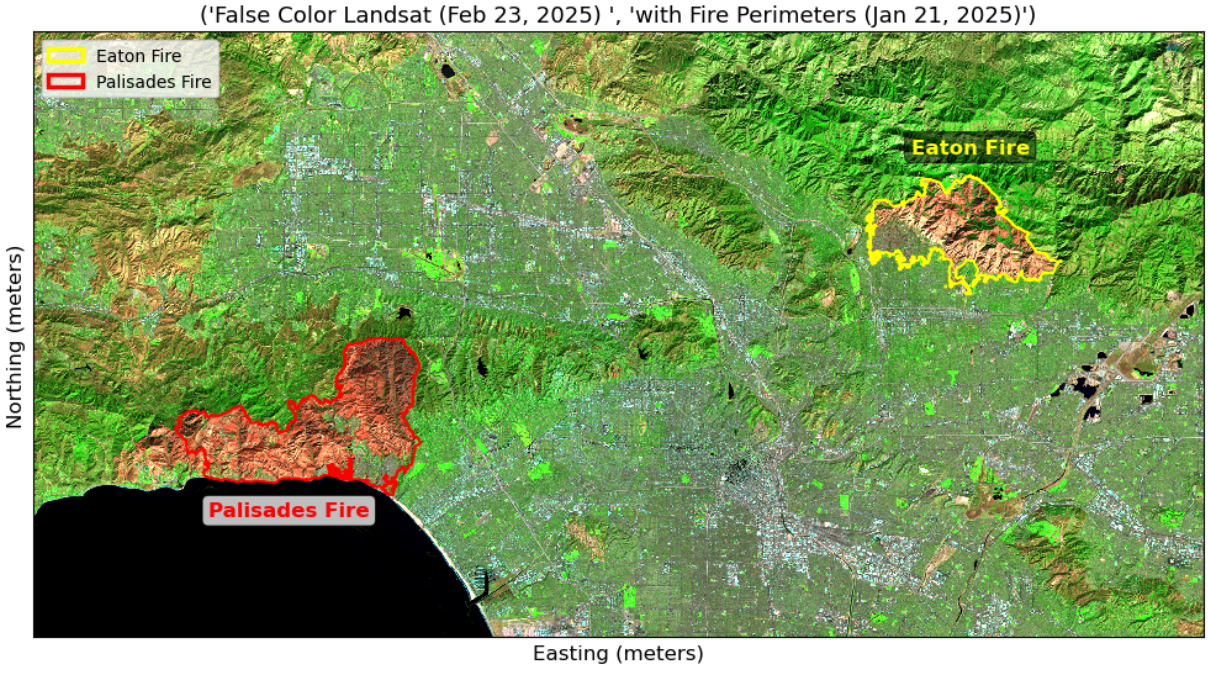
I will overlay the Eaton and Palisades fire boundaries on top of the false-color image and create a custom legend patches and labels to help identify each fire perimeter.

### a. Plot a map with false color with both fire perimeters

In [ ]:
```
# Create a figure and axes for plotting
fig, ax = plt.subplots(figsize = (10, 10))

# ----------------------------------------------------------------
# BASE LAYER
```

```
# ------------------------------------------------------------------

# Plot the false color landsat composite (SWIR2, NIR, Red)
(landsat_clean[['swir22', 'nir08', 'red']]
                                          .to_array()
                                          .plot.imshow(ax = ax,
                                                       add_colorbar = False,
                                                       robust = True))

# ------------------------------------------------------------------
# FIRE PERIMETER OVERLAYS
# ------------------------------------------------------------------

# Plot Eaton Fire perimeter using the raster's CRS
eaton.to_crs(landsat_clean.rio.crs).plot(ax = ax,
                                         edgecolor = 'yellow',
                                         facecolor = 'none',
                                         linewidth = 1.5)

# Plot Palisades Fire perimeter using the raster's CRS
palisades.to_crs(landsat_clean.rio.crs).plot(ax = ax,
                                         edgecolor = 'red',
                                         facecolor = 'none',
                                         linewidth = 1.5)

# ------------------------------------------------------------------
# CUSTOM LEGEND PATCHES
# ------------------------------------------------------------------

# Create a custom label bubble for Eaton
eaton_patch = mpatches.Patch(facecolor = 'none',
                             edgecolor = 'yellow',
                             linewidth = 2.5,
                             label = 'Eaton Fire')

# Create a custom label bubble for Palisades
palisades_patch = mpatches.Patch(facecolor = 'none',
                             edgecolor = 'red',
                             linewidth = 2.5,
                             label = 'Palisades Fire')

# Display a legend identifying each fire perimeter
ax.legend(handles = [eaton_patch, palisades_patch],
          loc = 'upper left')

# ------------------------------------------------------------------
# ON-MAP TEXT LABELS
# ------------------------------------------------------------------

# Eaton label (top right)
ax.text(0.75, 0.80,
        'Eaton Fire',
        transform = ax.transAxes,
        fontsize = 12,
        color = 'yellow',
        fontweight = 'bold',
```

```
        bbox = dict(boxstyle = 'round',
                    facecolor = 'black',
                    alpha = 0.5))

# Palisades label (bottom left)
ax.text(0.15, 0.20,
        'Palisades Fire',
        transform = ax.transAxes,
        fontsize = 12,
        color = 'red',
        fontweight = 'bold',
        bbox = dict(boxstyle = 'round',
                    facecolor = 'white',
                    alpha = 0.75))

# ------------------------------------------------------------------
# TITLE AND LAYOUT
# ------------------------------------------------------------------

# Label the map to clarify the image and overlays
ax.set_title(("False Color Landsat (Feb 23, 2025) ",
              "with Fire Perimeters (Jan 21, 2025)"),
              fontsize = 13)

# Remove both axis ticks
ax.set_xticks([])
ax.set_yticks([])

# Replace x and y label
ax.set_xlabel("Easting (meters)", fontsize = 12)
ax.set_ylabel("Northing (meters)", fontsize = 12)

# Improve layout so elements do not overlap
plt.tight_layout()
```



## b. Figure Description

This map shows a false color Landsat composite with SWIR2, NIR, Red bands over the perimeters for both Eaton and Palisade fires. In this false color image, shortwave infrared (SWIR2) and near-infraed(NIR) energy responses highlight differences between healthy vegetation, burned, and developed areas. Burned area appear in different color compare to healthy vegetation areas are bright green. Overlaying the fire perimeter boundaries on top of the images shows the visual comparison of burn patterns.