Problem_1

September 15, 2025

[2]:
```
pip install cmocean
```

```
Requirement already satisfied: cmocean in ./venv/lib/python3.13/site-packages
(4.0.3)
Requirement already satisfied: matplotlib in ./venv/lib/python3.13/site-packages
(from cmocean) (3.10.6)
Requirement already satisfied: numpy in ./venv/lib/python3.13/site-packages
(from cmocean) (2.3.2)
Requirement already satisfied: packaging in ./venv/lib/python3.13/site-packages
(from cmocean) (25.0)
Requirement already satisfied: contourpy>=1.0.1 in ./venv/lib/python3.13/site-
packages (from matplotlib->cmocean) (1.3.3)
Requirement already satisfied: cycler>=0.10 in ./venv/lib/python3.13/site-
packages (from matplotlib->cmocean) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in ./venv/lib/python3.13/site-
packages (from matplotlib->cmocean) (4.59.2)
Requirement already satisfied: kiwisolver>=1.3.1 in ./venv/lib/python3.13/site-
packages (from matplotlib->cmocean) (1.4.9)
Requirement already satisfied: pillow>=8 in ./venv/lib/python3.13/site-packages
(from matplotlib->cmocean) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in ./venv/lib/python3.13/site-
packages (from matplotlib->cmocean) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in
./venv/lib/python3.13/site-packages (from matplotlib->cmocean) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in ./venv/lib/python3.13/site-packages
(from python-dateutil>=2.7->matplotlib->cmocean) (1.17.0)

[notice] A new release of pip is
available: 25.1.1 -> 25.2
[notice] To update, run:
pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
import cmocean
from matplotlib.colors import LinearSegmentedColormap
import warnings
```

```
warnings.filterwarnings('ignore')
```

[ ]:
```
def load_sentinel2_data(file_path='/Users/varagantibasanthkumar/Desktop/Remote␣
 ↪sensing - IMGS 589/Homework1/sentinel2_rochester.npy'):
    """
    Load Sentinel-2 data and return the array

    This function reads the pre-processed Sentinel-2 satellite imagery data for␣
 ↪Rochester.
    The data has been saved as a NumPy array file (.npy) which contains surface␣
 ↪reflectance
    values for 12 different spectral bands.
    """
    # Load the NumPy array file containing the Sentinel-2 data
    # This file was extracted from a zip archive and contains the actual␣
 ↪satellite data
    data = np.load(file_path)

    # Print confirmation that the data loaded successfully
    print("Data loaded successfully!")

    # Display the dimensions of our dataset
    # Shape format: (height_pixels, width_pixels, number_of_bands)
    print("Shape: " + str(data.shape) + " (Height: " + str(data.shape[0]) + ",␣
 ↪Width: " + str(data.shape[1]) + ", Bands: " + str(data.shape[2]) + ")")

    # Show the data type - tells us how the numbers are stored in memory
    # float64 means 64-bit floating point numbers (good precision for␣
 ↪reflectance values)
    print("Data type: " + str(data.dtype))

    # Display the range of reflectance values in our dataset
    # These values represent surface reflectance (0 = no reflection, 1 =␣
 ↪perfect reflection)
    # Typical range for Sentinel-2 is 0 to 1, but can exceed 1 due to␣
 ↪atmospheric effects
    min_val = data.min()
    max_val = data.max()
    print("The reflectance values range from {:.4f} to {:.4f}".format(min_val,␣
 ↪max_val))

    # Return the loaded data array so other functions can use it
    return data
```

[ ]:
```
def get_sentinel2_band_info():
    """
```

```
    Return information about Sentinel-2 bands

    This function creates a dictionary containing metadata for each of the 12␣
↪spectral bands
    in our Sentinel-2 dataset. Each band captures different wavelengths of␣
↪light, which
    allows us to analyze various surface features and atmospheric conditions.

    Note: B10 (SWIR Cirrus) is missing from this dataset - this is common as␣
↪it's
    primarily used for atmospheric correction and cloud detection.
    """
    # Create a dictionary that maps band index to band information
    # This makes it easy to look up details about any band by its position in␣
↪the data array
    band_info = {
        # Band 0: Coastal Aerosol - used for atmospheric correction and coastal␣
↪water studies
        0: {'name': 'B1', 'description': 'Coastal Aerosol', 'wavelength': '443␣
↪nm', 'resolution': '60m'},

        # Band 1: Blue - visible light, good for water bodies and atmospheric␣
↪haze
        1: {'name': 'B2', 'description': 'Blue', 'wavelength': '490 nm',␣
↪'resolution': '10m'},

        # Band 2: Green - visible light, excellent for vegetation health␣
↪assessment
        2: {'name': 'B3', 'description': 'Green', 'wavelength': '560 nm',␣
↪'resolution': '10m'},

        # Band 3: Red - visible light, important for vegetation analysis and␣
↪soil studies
        3: {'name': 'B4', 'description': 'Red', 'wavelength': '665 nm',␣
↪'resolution': '10m'},

        # Band 4: Red Edge 1 - transition zone between red and near-infrared
        4: {'name': 'B5', 'description': 'Red Edge 1', 'wavelength': '705 nm',␣
↪'resolution': '20m'},

        # Band 5: Red Edge 2 - sensitive to chlorophyll content and vegetation␣
↪stress
        5: {'name': 'B6', 'description': 'Red Edge 2', 'wavelength': '740 nm',␣
↪'resolution': '20m'},

        # Band 6: Red Edge 3 - useful for vegetation monitoring and crop health
```

```
        6: {'name': 'B7', 'description': 'Red Edge 3', 'wavelength': '783 nm',␣
↪'resolution': '20m'},

        # Band 7: Near-Infrared (NIR) - most important for vegetation analysis
        7: {'name': 'B8', 'description': 'NIR', 'wavelength': '842 nm',␣
↪'resolution': '10m'},

        # Band 8: Red Edge 4 - narrow band for precise vegetation studies
        8: {'name': 'B8A', 'description': 'Red Edge 4', 'wavelength': '865 nm',␣
↪'resolution': '20m'},

        # Band 9: Water Vapor - atmospheric correction band, detects water␣
↪vapor content
        9: {'name': 'B9', 'description': 'Water Vapor', 'wavelength': '945 nm',␣
↪'resolution': '60m'},

        # Band 10: Short-Wave Infrared 1 - soil moisture, vegetation water␣
↪content
        10: {'name': 'B11', 'description': 'SWIR 1', 'wavelength': '1610 nm',␣
↪'resolution': '20m'},

        # Band 11: Short-Wave Infrared 2 - soil composition, mineral mapping
        11: {'name': 'B12', 'description': 'SWIR 2', 'wavelength': '2190 nm',␣
↪'resolution': '20m'}
    }

    # Return the complete band information dictionary
    return band_info
```

[ ]:
```
def identify_no_data(data, threshold=0.001):
    """
    Identify no-data pixels (very low values or NaN)

    This function finds pixels in the satellite imagery that don't contain␣
↪valid data.
    No-data pixels can occur due to clouds, shadows, sensor errors, or areas␣
↪outside
    the satellite's field of view. These pixels need to be identified and␣
↪handled
    properly during analysis.
    """
    # Create a boolean mask to identify problematic pixels
    # A mask is like a filter - True means "this pixel has no data", False␣
↪means "this pixel has valid data"
```

```
    # Check for pixels with values below our threshold (likely noise or invalid␣
↪data)
    # Sentinel-2 reflectance values should typically be between 0 and 1, so␣
↪values
    # below 0.001 are suspicious and probably represent no-data areas
    below_threshold = (data < threshold)

    # Check for NaN (Not a Number) values - these occur when calculations fail
    # or when the satellite sensor couldn't measure that pixel
    has_nan = np.isnan(data)

    # Combine both conditions using OR logic
    # A pixel is considered "no-data" if it's either below threshold OR␣
↪contains NaN
    no_data_mask = below_threshold | has_nan

    # Return the mask so other functions can use it to exclude bad pixels
    return no_data_mask
```

[ ]:
```
def apply_stretching(band_data, method='percentile', lower_percentile=2,␣
↪upper_percentile=98):
    """
    Apply stretching to improve visualization

    Raw satellite data often has poor contrast - most values are clustered in a␣
↪narrow range,
    making features hard to see. Stretching redistributes the pixel values to␣
↪use the full
    range of display colors, dramatically improving image contrast and detail␣
↪visibility.

    Methods: 'percentile', 'minmax', 'histogram_equalization'
    """
    if method == 'percentile':
        # Percentile-based stretching (most common for remote sensing)
        # This method ignores extreme outliers and focuses on the main data␣
↪distribution

        # Find the 2nd and 98th percentiles of valid data (excluding zero/
↪no-data pixels)
        # This removes the influence of extreme outliers that could skew our␣
↪stretching

        # We use band_data > 0 for BOTH calculations to exclude no-data pixels
        # No-data pixels (value 0) represent areas where the satellite couldn't␣
↪measure
```

```python
        # (clouds, shadows, sensor errors). Including them would skew our
↪percentiles
        # and make stretching ineffective. We only want to stretch actual
↪surface reflectance data.
        lower_bound = np.percentile(band_data[band_data > 0], lower_percentile)
        upper_bound = np.percentile(band_data[band_data > 0], upper_percentile)

        # Apply linear stretching: map the range [lower_bound, upper_bound] to
↪[0, 1]
        # Clip ensures no values go below 0 or above 1
        stretched = np.clip((band_data - lower_bound) / (upper_bound -
↪lower_bound), 0, 1)

    elif method == 'minmax':
        # Min-max stretching - uses the full range of actual data values
        # This method uses every pixel value, including outliers

        # Find the minimum and maximum values in valid data
        data_min = band_data[band_data > 0].min()
        data_max = band_data[band_data > 0].max()

        # Map the full data range to [0, 1]
        stretched = (band_data - data_min) / (data_max - data_min)

    elif method == 'histogram_equalization':
        # Histogram equalization - redistributes values to create uniform
↪histogram
        # This method tries to give equal "weight" to all brightness levels

        try:
            # Try to use scikit-image's histogram equalization function
            from skimage import exposure
            stretched = exposure.equalize_hist(band_data)
        except ImportError:
            # If scikit-image isn't available, fall back to percentile
↪stretching
            print("Warning: scikit-image not available, using percentile
↪stretching instead")
            lower_bound = np.percentile(band_data[band_data > 0], 2)
            upper_bound = np.percentile(band_data[band_data > 0], 98)
            stretched = np.clip((band_data - lower_bound) / (upper_bound -
↪lower_bound), 0, 1)

    # Return the stretched data ready for visualization
    return stretched
```

[ ]:
```
def plot_band(band_data, band_idx, band_info, fig, ax, colormap='viridis'):
    """
    Plot a single Sentinel-2 band with proper visualization

    This function takes raw satellite band data and creates a␣
↪publication-quality
    visualization. It handles all the necessary preprocessing steps including
    no-data masking, contrast stretching, and proper labeling.
    """
    # Extract band metadata from our information dictionary
    # This gives us the human-readable details to display on our plot
    band_name = band_info[band_idx]['name']          # e.g., "B2", "B8", "B11"
    band_desc = band_info[band_idx]['description']  # e.g., "Blue", "NIR",␣
↪"SWIR 1"
    wavelength = band_info[band_idx]['wavelength']   # e.g., "490 nm", "842 nm"
    resolution = band_info[band_idx]['resolution']  # e.g., "10m", "20m", "60m"

    # Identify problematic pixels that don't contain valid data
    # This creates a boolean mask marking pixels as "good" or "bad"
    no_data_mask = identify_no_data(band_data)

    # Apply contrast stretching to improve image visibility
    # Raw satellite data often has poor contrast - stretching fixes this
    stretched_data = apply_stretching(band_data, method='percentile')

    # Create a working copy of the stretched data for visualization
    # We don't want to modify the original stretched data
    vis_data = stretched_data.copy()

    # Handle no-data pixels by setting them to NaN (Not a Number)
    # NaN values appear transparent in matplotlib, so no-data areas won't show␣
↪up
    # This prevents confusing artifacts from appearing in our visualization
    vis_data[no_data_mask] = np.nan

    # Create the actual image plot
    # aspect='equal' ensures pixels are square (no distortion)
    im = ax.imshow(vis_data, cmap=colormap, aspect='equal')

    # Create an informative title combining all band information
    # This helps viewers understand exactly what they're looking at
    title = band_name + ": " + band_desc + "\n" + wavelength + " (" +␣
↪resolution + ")"
    ax.set_title(title, fontsize=10, fontweight='bold')

    # Remove axis ticks and labels for cleaner appearance
    # Satellite imagery doesn't need coordinate axes in most cases
```

```
    ax.axis('off')

    # Add a colorbar to show the mapping between colors and reflectance values
    # shrink=0.8 makes it slightly smaller to fit better in the subplot
    cbar = plt.colorbar(im, ax=ax, shrink=0.8)
    cbar.set_label('Reflectance', fontsize=8)

    # Return the image object in case other functions need to reference it
    return im
```

[9]:

```
def plot_all_bands(data, output_file='sentinel2_all_bands.png'):
    """
    Plot all 12 Sentinel-2 bands in a 4x3 grid
    """
    band_info = get_sentinel2_band_info()

    # Create figure with subplots
    fig, axes = plt.subplots(4, 3, figsize=(18, 24))
    fig.suptitle('Sentinel-2 Rochester Dataset - All 12 Bands\nSurface␣
↪Reflectance Data (Resampled to 30m)',
                 fontsize=16, fontweight='bold')

    # Flatten axes array for easier indexing
    axes_flat = axes.flatten()

    # Plot each band
    for i in range(12):
        band_data = data[:, :, i]
        plot_band(band_data, i, band_info, fig, axes_flat[i],␣
↪colormap='viridis')

    # Adjust layout
    plt.tight_layout()
    plt.subplots_adjust(top=0.95)

    # Save the plot
    plt.savefig(output_file, dpi=300, bbox_inches='tight')
    print("All bands plot saved as: " + output_file)

    plt.show()
```

[10]:

```
def plot_bands_with_different_colormaps(data, output_file='sentinel2_colormaps.
↪png'):
    """
    Plot bands using different vibrant colormaps from cmocean
    """
    band_info = get_sentinel2_band_info()
```

```python
# Define colormaps for different band types
colormaps = {
    'visible': 'viridis',       # Blue, Green, Red
    'red_edge': 'magma',        # Red edge bands
    'nir': 'plasma',            # NIR
    'swir': 'inferno',          # SWIR bands
    'atmospheric': cmocean.cm.thermal  # Atmospheric correction bands
}

# Assign colormaps to bands (12 bands total, B10 missing from dataset)
band_colormaps = [
    colormaps['atmospheric'],  # B1 - Coastal Aerosol
    colormaps['visible'],      # B2 - Blue
    colormaps['visible'],      # B3 - Green
    colormaps['visible'],      # B4 - Red
    colormaps['red_edge'],     # B5 - Red Edge 1
    colormaps['red_edge'],     # B6 - Red Edge 2
    colormaps['red_edge'],     # B7 - Red Edge 3
    colormaps['nir'],          # B8 - NIR
    colormaps['red_edge'],     # B8A - Red Edge 4
    colormaps['atmospheric'],  # B9 - Water Vapor
    colormaps['swir'],         # B11 - SWIR 1
    colormaps['swir']          # B12 - SWIR 2
]

# Create figure
fig, axes = plt.subplots(4, 3, figsize=(18, 24))
fig.suptitle('Sentinel-2 Rochester Dataset - All 12 Bands with Vibrant␣
↪Colormaps\nSurface Reflectance Data',
             fontsize=16, fontweight='bold')

axes_flat = axes.flatten()

# Plot each band with appropriate colormap
for i in range(12):
    band_data = data[:, :, i]
    colormap = band_colormaps[i]
    plot_band(band_data, i, band_info, fig, axes_flat[i], colormap=colormap)

plt.tight_layout()
plt.subplots_adjust(top=0.95)

# Save the plot
plt.savefig(output_file, dpi=300, bbox_inches='tight')
print("Colormap plot saved as: " + output_file)
```

```python
    plt.show()
```

[11]:
```python
def analyze_no_data(data):
    """
    Analyze the no-data regions in the dataset
    """
    print("\n=== NO-DATA ANALYSIS ===")

    # Identify no-data pixels
    no_data_mask = identify_no_data(data)

    # Calculate statistics
    total_pixels = data.shape[0] * data.shape[1]
    no_data_pixels = np.sum(no_data_mask, axis=(0, 1))
    no_data_percentage = (no_data_pixels / total_pixels) * 100

    print("Total pixels per band: " + str(total_pixels))
    print("No-data pixels per band:")

    band_info = get_sentinel2_band_info()
    for i in range(12):
        band_name = band_info[i]['name']
        print("  " + band_name + ": " + str(no_data_pixels[i]) + " pixels (" +␣
 ↪"{:.2f}".format(no_data_percentage[i]) + "%)")

    # Create a composite no-data mask
    composite_no_data = np.any(no_data_mask, axis=2)
    composite_percentage = (np.sum(composite_no_data) / total_pixels) * 100
    print("\nComposite no-data: " + str(np.sum(composite_no_data)) + " pixels␣
 ↪(" + "{:.2f}".format(composite_percentage) + "%)")

    return no_data_mask, composite_no_data
```

[12]:
```python
def main():
    """
    Main function to execute the complete analysis
    """
    print("=== SENTINEL-2 ROCHESTER DATASET ANALYSIS ===")
    print("Problem 1: Band Visualization and Analysis\n")

    # Load data
    data = load_sentinel2_data()

    # Analyze no-data regions
    no_data_mask, composite_no_data = analyze_no_data(data)

    # Plot all bands with default colormap
```

```
    print("\n=== GENERATING PLOTS ===")
    plot_all_bands(data)

    # Plot with different colormaps
    plot_bands_with_different_colormaps(data)

    # Additional analysis
    print("\n=== ADDITIONAL ANALYSIS ===")
    print("1. No-data handling approach:")
    print("   - Identified using threshold-based approach")
    print("   - Set to NaN for transparency in visualization")
    print("   - This approach will be reused in subsequent problems")

    print("\n2. Visualization stretching approach:")
    print("   - Used percentile-based stretching (2-98%)")
    print("   - Excludes no-data pixels from percentile calculation")
    print("   - Provides better contrast while preserving data integrity")

    print("\n3. Colormap selection:")
    print("   - Used vibrant colormaps from cmocean library")
    print("   - Different colormaps for different band types")
    print("   - Visible bands: viridis")
    print("   - Red edge bands: magma")
    print("   - NIR: plasma")
    print("   - SWIR: inferno")
    print("   - Atmospheric: cmo.thermal")

if __name__ == "__main__":
    main()
```

```
=== SENTINEL-2 ROCHESTER DATASET ANALYSIS ===
Problem 1: Band Visualization and Analysis
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[12], line 43
     40     print("   - Atmospheric: cmo.thermal")
     42 if __name__ == "__main__":
---> 43     main()

Cell In[12], line 9, in main()
      6 print("Problem 1: Band Visualization and Analysis\n")
      8 # Load data
----> 9 data = load_sentinel2_data()
     11 # Analyze no-data regions
     12 no_data_mask, composite_no_data = analyze_no_data(data)
```

```
Cell In[4], line 5, in load_sentinel2_data(file_path)
      1 def load_sentinel2_data(file_path='sentinel2_rochester.npy'):
      2     """
      3     Load Sentinel-2 data and return the array
      4     """
----> 5     data = np.load(file_path)
      6     print("Data loaded successfully!")
      7     print("Shape: " + str(data.shape) + " (Height: " + str(data.
 ↪shape[0]) + ", Width: " + str(data.shape[1]) + ", Bands: " + str(data.
 ↪shape[2]) + ")")

File ~/Desktop/Remote sensing - IMGS 589/Homework1/venv/lib/python3.13/
 ↪site-packages/numpy/lib/_npyio_impl.py:454, in load(file, mmap_mode,␣
 ↪allow_pickle, fix_imports, encoding, max_header_size)
    452     own_fid = False
    453 else:
--> 454     fid = stack.enter_context(open(os.fspath(file),      ))
    455     own_fid = True
    457 # Code to distinguish from NumPy binary files and pickles.

FileNotFoundError: [Errno 2] No such file or directory: 'sentinel2_rochester.np '
```

```
[ ]:
```