Ranjeet Gupta / SC24M138

In [2]:
```
!pip install rasterio
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting rasterio
  Downloading rasterio-1.3.11-cp312-cp312-win_amd64.whl.metadata (15 kB)
Collecting affine (from rasterio)
  Downloading affine-2.4.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: attrs in c:\programdata\anaconda3\lib\site-package
s (from rasterio) (23.1.0)
Requirement already satisfied: certifi in c:\programdata\anaconda3\lib\site-packa
ges (from rasterio) (2024.6.2)
Requirement already satisfied: click>=4.0 in c:\programdata\anaconda3\lib\site-pa
ckages (from rasterio) (8.1.7)
Collecting cligj>=0.5 (from rasterio)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-package
s (from rasterio) (1.26.4)
Collecting snuggs>=1.4.1 (from rasterio)
  Downloading snuggs-1.4.7-py3-none-any.whl.metadata (3.4 kB)
Collecting click-plugins (from rasterio)
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl.metadata (6.4 kB)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\lib\site-pa
ckages (from rasterio) (69.5.1)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-pack
ages (from click>=4.0->rasterio) (0.4.6)
Requirement already satisfied: pyparsing>=2.1.6 in c:\programdata\anaconda3\lib\s
ite-packages (from snuggs>=1.4.1->rasterio) (3.0.9)
Downloading rasterio-1.3.11-cp312-cp312-win_amd64.whl (24.8 MB)
   ---------------------------------------- 0.0/24.8 MB ? eta -:--:--
   ---------------------------------------- 0.0/24.8 MB ? eta -:--:--
   ---------------------------------------- 0.1/24.8 MB 975.2 kB/s eta 0:00:26
   ---------------------------------------- 0.1/24.8 MB 1.1 MB/s eta 0:00:23
   ---------------------------------------- 0.2/24.8 MB 1.4 MB/s eta 0:00:18
    --------------------------------------- 0.3/24.8 MB 1.6 MB/s eta 0:00:15
    --------------------------------------- 0.5/24.8 MB 1.8 MB/s eta 0:00:14
    --------------------------------------- 0.6/24.8 MB 1.9 MB/s eta 0:00:13
   - --------------------------------------- 0.7/24.8 MB 1.8 MB/s eta 0:00:14
   - --------------------------------------- 0.7/24.8 MB 1.8 MB/s eta 0:00:14
   - --------------------------------------- 0.8/24.8 MB 1.8 MB/s eta 0:00:14
   - --------------------------------------- 0.9/24.8 MB 1.9 MB/s eta 0:00:13
   - --------------------------------------- 1.0/24.8 MB 1.8 MB/s eta 0:00:14
   - --------------------------------------- 1.1/24.8 MB 1.8 MB/s eta 0:00:14
   - --------------------------------------- 1.1/24.8 MB 1.8 MB/s eta 0:00:14
   - --------------------------------------- 1.2/24.8 MB 1.8 MB/s eta 0:00:14
   -- -------------------------------------- 1.2/24.8 MB 1.7 MB/s eta 0:00:14
   -- -------------------------------------- 1.3/24.8 MB 1.7 MB/s eta 0:00:14
   -- -------------------------------------- 1.4/24.8 MB 1.7 MB/s eta 0:00:14
   -- -------------------------------------- 1.4/24.8 MB 1.7 MB/s eta 0:00:14
   -- -------------------------------------- 1.5/24.8 MB 1.6 MB/s eta 0:00:15
   -- -------------------------------------- 1.5/24.8 MB 1.6 MB/s eta 0:00:15
   -- -------------------------------------- 1.6/24.8 MB 1.6 MB/s eta 0:00:15
   -- -------------------------------------- 1.6/24.8 MB 1.5 MB/s eta 0:00:16
   -- -------------------------------------- 1.6/24.8 MB 1.5 MB/s eta 0:00:16
   -- -------------------------------------- 1.7/24.8 MB 1.4 MB/s eta 0:00:16
   -- -------------------------------------- 1.7/24.8 MB 1.4 MB/s eta 0:00:16
   -- -------------------------------------- 1.7/24.8 MB 1.4 MB/s eta 0:00:17
   -- -------------------------------------- 1.8/24.8 MB 1.4 MB/s eta 0:00:17
   --- ------------------------------------- 1.9/24.8 MB 1.4 MB/s eta 0:00:16
   --- ------------------------------------- 2.0/24.8 MB 1.4 MB/s eta 0:00:16
   --- ------------------------------------- 2.0/24.8 MB 1.4 MB/s eta 0:00:17
   --- ------------------------------------- 2.1/24.8 MB 1.4 MB/s eta 0:00:16
   --- ------------------------------------- 2.2/24.8 MB 1.5 MB/s eta 0:00:16
```

```
   ---------------------------------- ----- 21.5/24.8 MB 1.1 MB/s eta 0:00:03
   ---------------------------------- ----- 21.5/24.8 MB 1.0 MB/s eta 0:00:04
   ---------------------------------- ---- 21.9/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 21.9/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 21.9/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 21.9/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 22.0/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 22.1/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 22.1/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 22.2/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 22.2/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 22.2/24.8 MB 1.0 MB/s eta 0:00:03
   ---------------------------------- ---- 22.2/24.8 MB 991.9 kB/s eta 0:00:03
   ---------------------------------- ---- 22.2/24.8 MB 991.9 kB/s eta 0:00:03
   ---------------------------------- ---- 22.2/24.8 MB 991.9 kB/s eta 0:00:03
   ---------------------------------- ---- 22.2/24.8 MB 978.6 kB/s eta 0:00:03
   ---------------------------------- ---- 22.3/24.8 MB 978.6 kB/s eta 0:00:03
   ---------------------------------- ---- 22.3/24.8 MB 975.7 kB/s eta 0:00:03
   ----------------------------------- --- 22.3/24.8 MB 972.8 kB/s eta 0:00:03
   ----------------------------------- --- 22.3/24.8 MB 965.6 kB/s eta 0:00:03
   ----------------------------------- --- 22.3/24.8 MB 965.6 kB/s eta 0:00:03
   ----------------------------------- --- 22.3/24.8 MB 957.2 kB/s eta 0:00:03
   ----------------------------------- --- 22.3/24.8 MB 954.4 kB/s eta 0:00:03
   ----------------------------------- --- 22.3/24.8 MB 954.4 kB/s eta 0:00:03
   ----------------------------------- --- 22.4/24.8 MB 951.6 kB/s eta 0:00:03
   ----------------------------------- --- 22.4/24.8 MB 947.5 kB/s eta 0:00:03
   ----------------------------------- --- 22.4/24.8 MB 944.7 kB/s eta 0:00:03
   ----------------------------------- --- 22.5/24.8 MB 946.1 kB/s eta 0:00:03
   ----------------------------------- --- 22.5/24.8 MB 943.4 kB/s eta 0:00:03
   ----------------------------------- --- 22.6/24.8 MB 939.3 kB/s eta 0:00:03
   ----------------------------------- --- 22.6/24.8 MB 943.4 kB/s eta 0:00:03
   ----------------------------------- --- 22.7/24.8 MB 943.4 kB/s eta 0:00:03
   ----------------------------------- --- 22.8/24.8 MB 946.1 kB/s eta 0:00:03
   ----------------------------------- --- 22.9/24.8 MB 954.4 kB/s eta 0:00:02
   ------------------------------------ -- 22.9/24.8 MB 968.5 kB/s eta 0:00:02
   ------------------------------------ -- 23.1/24.8 MB 977.1 kB/s eta 0:00:02
   ------------------------------------ -- 23.2/24.8 MB 989.0 kB/s eta 0:00:02
   ------------------------------------ -- 23.3/24.8 MB 993.5 kB/s eta 0:00:02
   ------------------------------------ -- 23.4/24.8 MB 992.0 kB/s eta 0:00:02
   ------------------------------------ -- 23.5/24.8 MB 996.5 kB/s eta 0:00:02
   ------------------------------------- - 23.6/24.8 MB 998.0 kB/s eta 0:00:02
   ------------------------------------- - 23.7/24.8 MB 996.5 kB/s eta 0:00:02
   ------------------------------------- - 23.8/24.8 MB 998.0 kB/s eta 0:00:01
   ------------------------------------- - 23.8/24.8 MB 996.5 kB/s eta 0:00:01
   ------------------------------------- - 23.9/24.8 MB 996.6 kB/s eta 0:00:01
   ------------------------------------- - 24.0/24.8 MB 995.0 kB/s eta 0:00:01
   ------------------------------------- - 24.0/24.8 MB 995.0 kB/s eta 0:00:01
   --------------------------------------  24.2/24.8 MB 999.6 kB/s eta 0:00:01
   --------------------------------------  24.2/24.8 MB 999.6 kB/s eta 0:00:01
   --------------------------------------  24.3/24.8 MB 996.5 kB/s eta 0:00:01
   --------------------------------------  24.4/24.8 MB 996.5 kB/s eta 0:00:01
   --------------------------------------  24.5/24.8 MB 995.0 kB/s eta 0:00:01
   --------------------------------------  24.5/24.8 MB 1.0 MB/s eta 0:00:01
   --------------------------------------  24.6/24.8 MB 1.0 MB/s eta 0:00:01
   --------------------------------------  24.7/24.8 MB 1.0 MB/s eta 0:00:01
   --------------------------------------  24.8/24.8 MB 1.0 MB/s eta 0:00:01
   --------------------------------------  24.8/24.8 MB 1.0 MB/s eta 0:00:01
   --------------------------------------- 24.8/24.8 MB 1.0 MB/s eta 0:00:00
Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Downloading snuggs-1.4.7-py3-none-any.whl (5.4 kB)
```

```
Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Installing collected packages: snuggs, affine, cligj, click-plugins, rasterio
Successfully installed affine-2.4.0 click-plugins-1.1.1 cligj-0.7.2 rasterio-1.3.
11 snuggs-1.4.7
```

```
  WARNING: The script rio.exe is installed in 'C:\Users\Ranjeet Gupta\AppData\Roa
ming\Python\Python312\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warni
ng, use --no-warn-script-location.
```

1) Download Sentinel data for your city

In [ ]:

2) Read the raster file and gather basic information (dimension of data, number of bands , spatial resolution, projection system)

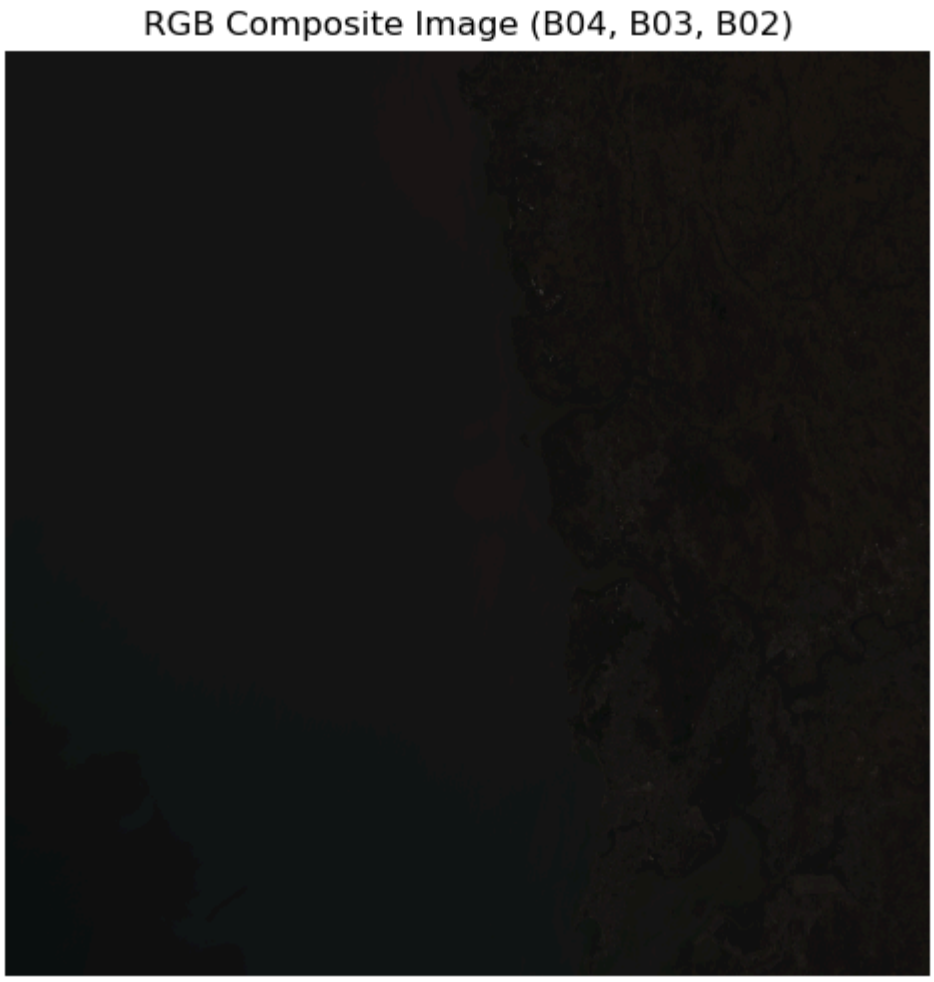
In [4]:
```
import rasterio

dataset = rasterio.open(r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T0

print("Dimensions (Width x Height):", dataset.width, "", dataset.height)
print("Number of Bands:", dataset.count)
print("Spatial Resolution:", dataset.res)
print("Coordinate Reference System (CRS):", dataset.crs)
```

```
Dimensions (Width x Height): 10980  10980
Number of Bands: 3
Spatial Resolution: (10.0, 10.0)
Coordinate Reference System (CRS): EPSG:32643
```

3. Stack the individual bands and form a single image file

In [6]:
```
import rasterio
from rasterio.merge import merge
from rasterio.plot import show
import numpy as np
import matplotlib.pyplot as plt


# Paths to your individual band files
band1_file = r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0510
band2_file = r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0510
band3_file = r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0510

# Open the individual bands
with rasterio.open(band1_file) as band1, rasterio.open(band2_file) as band2, ras
```

```
    # Read the data from each band
    band1_data = band1.read(1)  # Reads the first band (assuming it's a single-b
    band2_data = band2.read(1)
    band3_data = band3.read(1)

    # Stack the bands into a single array
    stacked_array = np.dstack((band1_data, band2_data, band3_data)) #Stacks the

    # Normalize pixel values to the range 0-255 (this is needed for proper RGB d
    def normalize(array):
        array_min, array_max = array.min(), array.max()
        return ((array - array_min) / (array_max - array_min) * 255).astype(np.u

    merged_array = normalize(stacked_array)

    # Display the normalized image using matplotlib
    plt.figure(figsize=(6, 6))
    plt.imshow(merged_array)
    plt.title('RGB Composite Image (B04, B03, B02)')
    plt.axis('off')  # Hide axes for a cleaner view
    plt.show()

    print("Stacked image displayed successfully.")

#     # mosaic, out_transform = merge([band1, band2, band3])  #to combine them i

#     # Get metadata from the first band
#     out_meta = band1.meta.copy()

#     # Update metadata for the new multi-band file
#     out_meta.update({"count": 3})  # Set the number of bands to 3

#     # out_meta.update({
#     #     'driver': 'GTiff',
#     #     'height': mosaic.shape[1],
#     #     'width': mosaic.shape[2],
#     #     'transform': out_transform,
#     #     "count": 3})

#     # Write the stacked bands to a new file
#     with rasterio.open('stacked_image.tif', 'w', **out_meta) as dest:
#         dest.write(stacked_array)

#     # Open the new stacked image for display
#     with rasterio.open('stacked_image.tif') as stacked_dataset:
#         show(stacked_dataset)

# print("Stacked image file created and displayed successfully.")
```



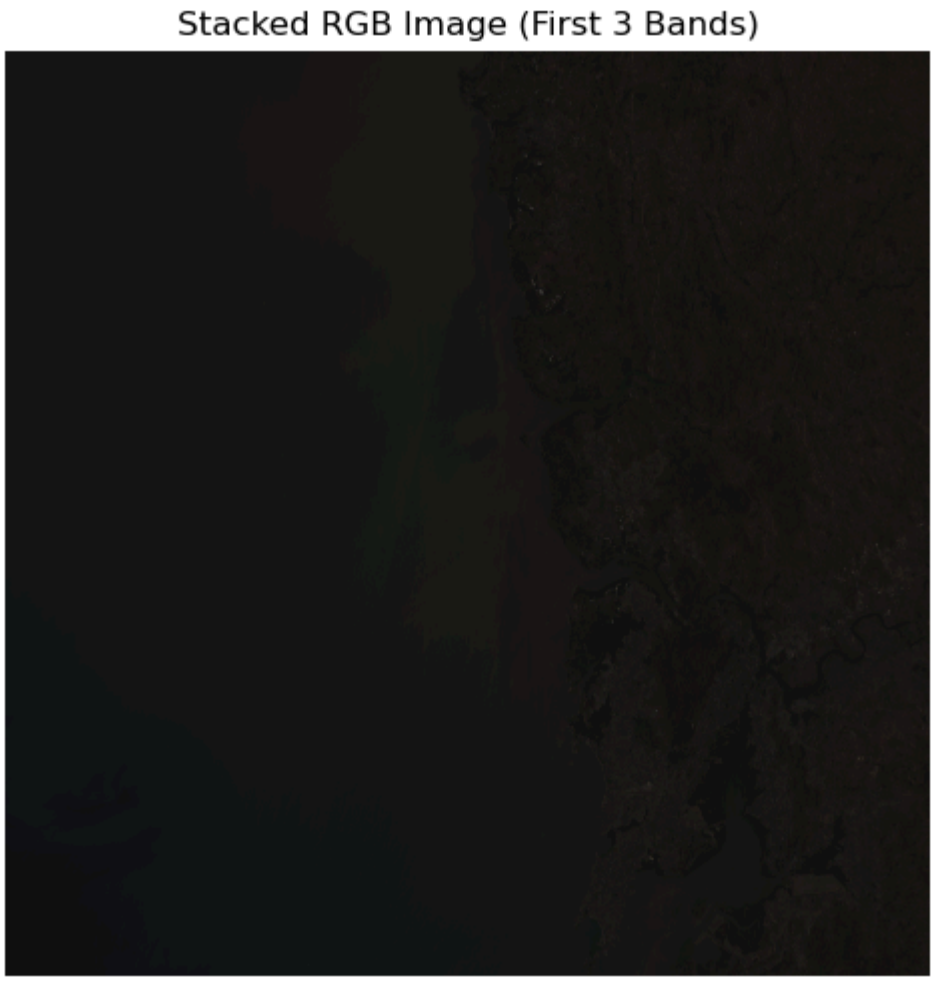

```
Stacked image displayed successfully.
```

4) Create a function to create stack of bands which will take input of bands and provide stacked output.

In [55]:
```python
import rasterio
import numpy as np
import matplotlib.pyplot as plt

def stack_bands_withdisplay(band_files, output_file):
    """
    Function to stack multiple raster bands into a single image.
    """

    # Open the first band to get the metadata and dimensions
    with rasterio.open(band_files[0]) as src:
        # Read the first band's data and metadata
        meta = src.meta
        # Create an empty list to hold band data
        band_data = [src.read(1)]

    # Read the remaining bands and append them to the band_data list
    for band in band_files[1:]:
        with rasterio.open(band) as src:
            band_data.append(src.read(1))
```

```
    # Stack the bands into a 3D array (along the first axis)
    stacked_array = np.stack(band_data, axis=0)

    # Update the metadata to reflect the number of bands
    meta.update(count=len(band_files))

    # Write the stacked array to the output file
    with rasterio.open(output_file, 'w', **meta) as dest:
        dest.write(stacked_array)

    print(f"Stacked image saved as: {output_file}")

    # Normalize the data for display (assuming the data is in 16-bit format)
    stacked_normalized = stacked_array.astype(np.float32) / np.max(stacked_array

    # Display the first three bands as an RGB image if we have at least 3 bands
    if len(band_files) >= 3:
        # Stack the first three bands into an RGB composite
        rgb_image = np.dstack((stacked_normalized[0], stacked_normalized[1], sta

        # Display the RGB image using matplotlib
        plt.figure(figsize=(6, 6))
        plt.imshow(rgb_image)
        plt.title('Stacked RGB Image (First 3 Bands)')
        plt.axis('off')  # Turn off the axis labels
        plt.show()
    else:
        print("Not enough bands for RGB display. At least 3 bands are needed.")

    return stacked_array

band_files = [band1_file, band2_file, band3_file]  # as previous cell declared

# Call the function to stack the bands and save the output
stacked_array = stack_bands_withdisplay(band_files, output_file='stacked_output.
```

```
Stacked image saved as: stacked_output.tif
```



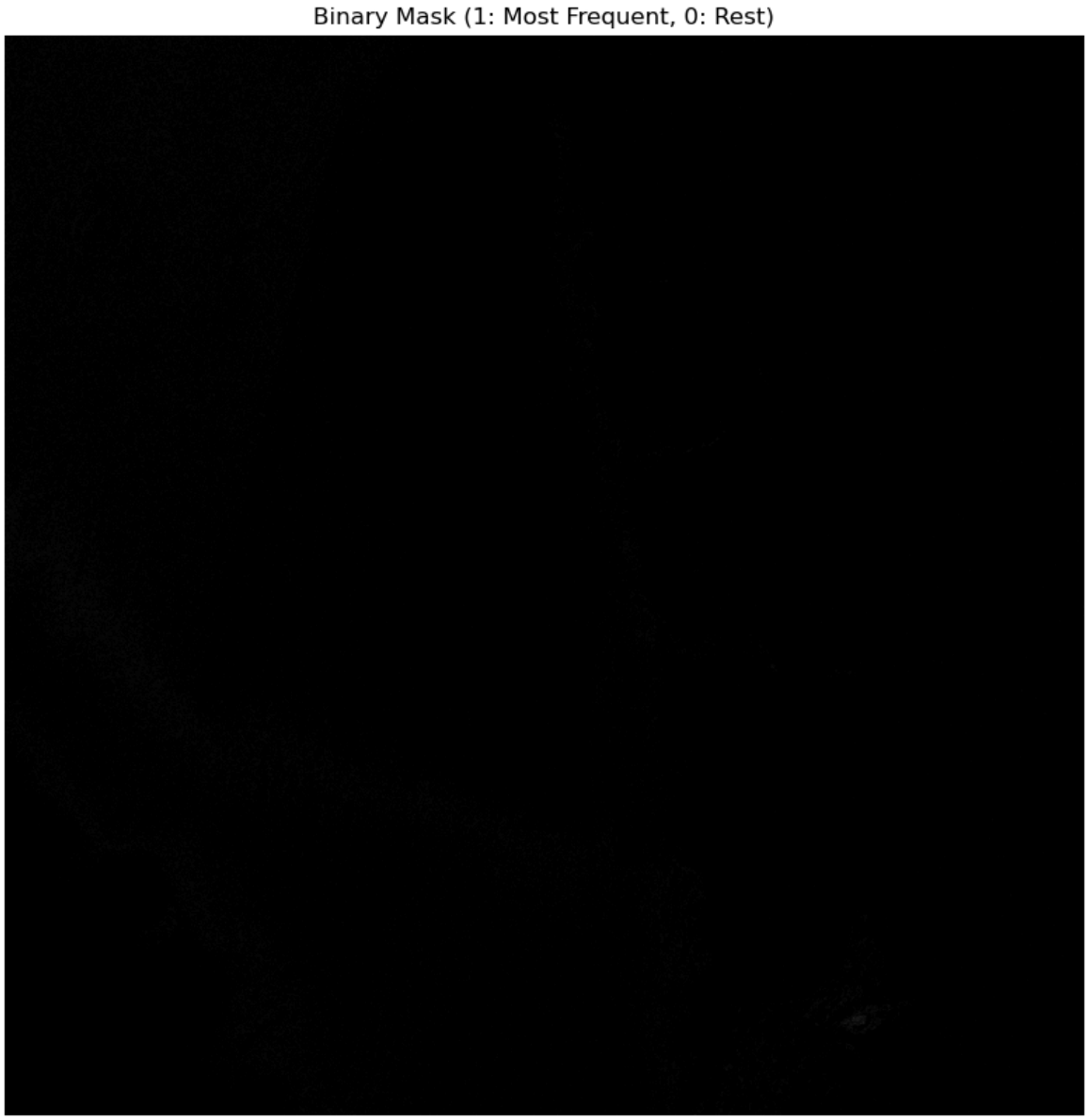
5) Find out the most repeating value in the stacked image and generate binary mask which includes two classes, most repeating value and rest of the values. Copy location information(extent) from stacked image to this masked output.

In [27]:

```
import rasterio
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

def generate_binary_mask(stacked_band_files, output_file='binary_mask_output.tif
    """
    Function to create a binary mask based on the most repeating value in a stac
    Returns:
    - None: The function writes the binary mask to the output file and displays
    """
    with rasterio.open(stacked_band_files) as src:

        stacked_array = src.read() # Read all bands into a 3D array (band, row,
        meta = src.meta  # Get the metadata to retain location information

    # Flatten the stacked array to a 1D array to find the most frequent value
```

```
    flattened = stacked_array.flatten()

    # Find the most repeating pixel value using numpy's bincount and argmax
    most_frequent_value = np.bincount(flattened).argmax()
    print(f"Most repeating value in the stacked image: {most_frequent_value}")

    # Create a binary mask (1 for the most frequent value, 0 for everything else
    binary_mask = (stacked_array == most_frequent_value).astype(np.uint8)

    # Reduce the mask to a single 2D layer by applying across all bands using np
    binary_mask_combined = np.any(binary_mask, axis=0).astype(np.uint8)

    # Update metadata for a single-band mask image
    meta.update(count=1, dtype='uint8')  # Change driver to GeoTIFF

    # Write the binary mask to the output file
    with rasterio.open(output_file, 'w', **meta) as dest:
        dest.write(binary_mask_combined, 1)  # Write the binary mask as a single

    print(f"Binary mask saved as: {output_file}")

    # Display the binary mask using matplotlib
    plt.figure(figsize=(10, 10))
    plt.imshow(binary_mask_combined, cmap='gray')
    plt.title('Binary Mask (1: Most Frequent, 0: Rest)')
    plt.axis('off')
    plt.show()

stacked_band_files = r"C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab
generate_binary_mask(stacked_band_files, output_file='binary_mask_output.tif')
```

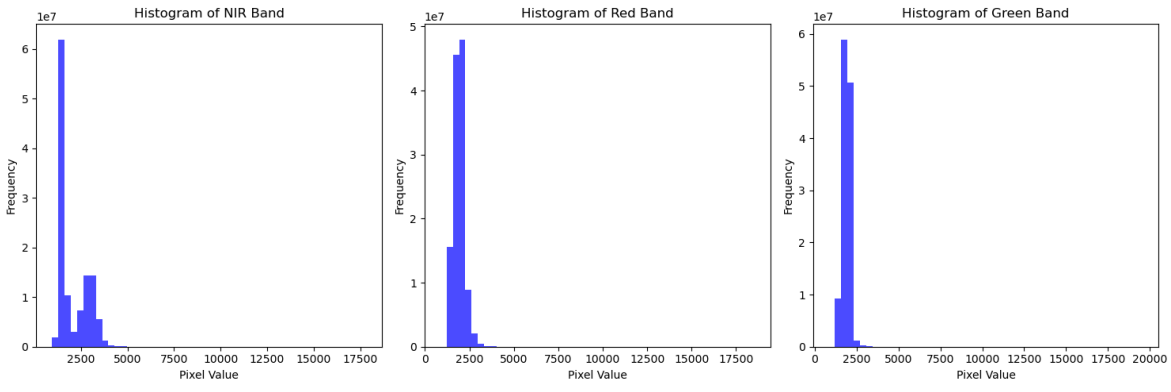
```
Most repeating value in the stacked image: 1810
Binary mask saved as: binary_mask_output.tif
```



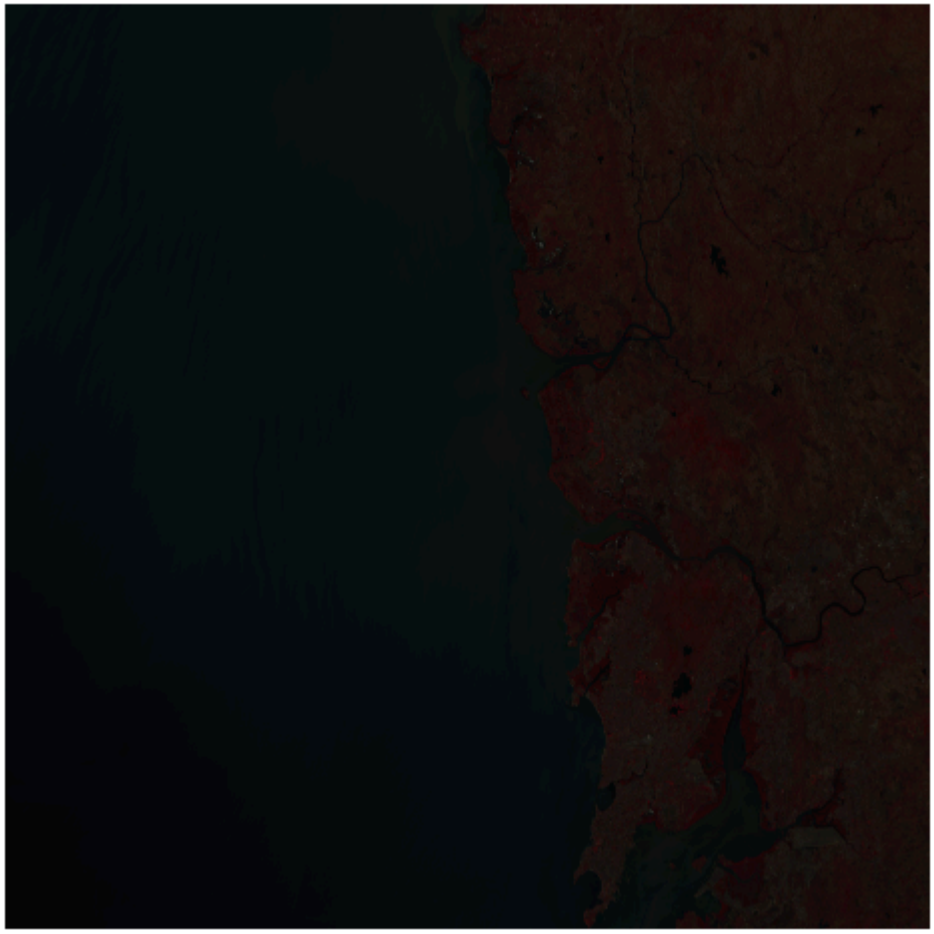
6) Display the histogram of each band and create a false color composite

In [35]:
```
import rasterio
import numpy as np
import matplotlib.pyplot as plt

band_files = [
    r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0510_R005_T43
    r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0510_R005_T43
    r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0510_R005_T43
]

# Initialize an empty list to hold the band data
band_data = []

band_names = ['NIR', 'Red', 'Green']

# Create a figure for subplots
fig, axes = plt.subplots(1, len(band_files), figsize=(15, 5), tight_layout=True)
```

```python
# Iterate through each band file, read it, and append the data to band_data list
for idx, band_file in enumerate(band_files):
    with rasterio.open(band_file) as src:
        band = src.read(1)  # Read the band data
        band_data.append(band)

        # Display histogram for the current band in a subplot
        axes[idx].hist(band.flatten(), bins=50, color='blue', alpha=0.7)
        axes[idx].set_title(f'Histogram of {band_names[idx]} Band')
        axes[idx].set_xlabel('Pixel Value')
        axes[idx].set_ylabel('Frequency')

# Show the histogram subplot figure
plt.show()

# Stack the bands into a 3D array (for FCC creation)
stacked_array = np.stack(band_data, axis=0)

# Create a False Color Composite (FCC)
if len(band_files) >= 3:
    # Typically, FCC uses: NIR (near-infrared) for Red, Red for Green, and Green
    false_color_composite = np.dstack((stacked_array[0], stacked_array[1], stack

    # Normalize for display (simple min-max normalization)
    false_color_composite = false_color_composite.astype(np.float32)
    for i in range(3):
        min_val, max_val = false_color_composite[:, :, i].min(), false_color_com
        false_color_composite[:, :, i] = (false_color_composite[:, :, i] - min_v

    # Display the FCC image
    plt.figure(figsize=(6, 6))
    plt.imshow(false_color_composite)
    plt.title('False Color Composite (FCC)')
    plt.axis('off')
    plt.show()
else:
    print("At least 3 bands are required to create a False Color Composite (FCC)
```



False Color Composite (FCC)



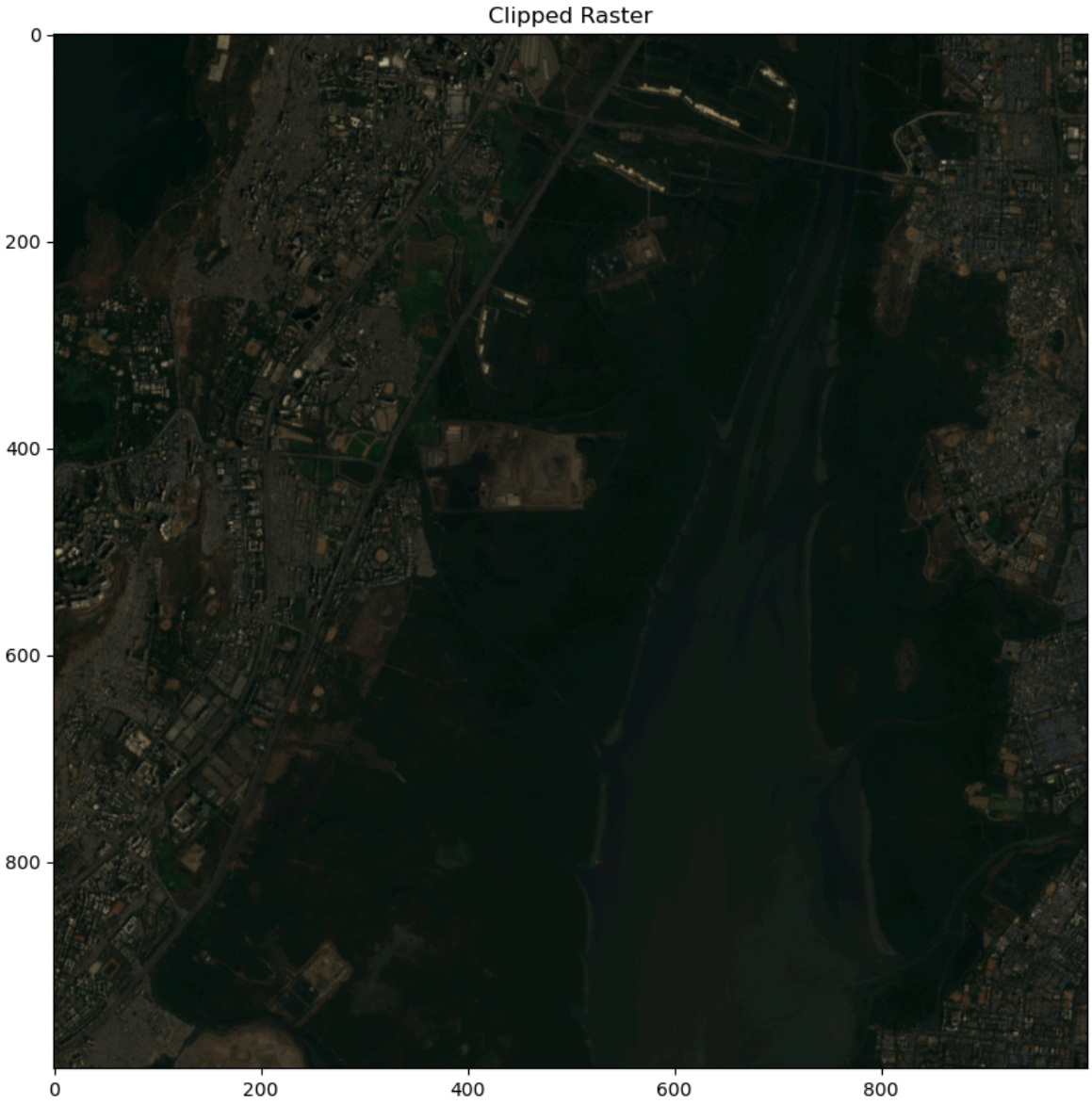
7) Clip and save a subset of data

In [61]:

```
import rasterio
from rasterio.windows import Window
import numpy as np
import matplotlib.pyplot as plt

def clip_and_save_raster(input_raster, output_raster, window_bounds):
    """
    Parameters:
    - window_bounds: The window coordinates (row_start, col_start, height, width
    Returns:
    - None: The function writes the clipped raster to the output file and displa
    """
    # Open the input raster
    with rasterio.open(input_raster) as src:
        # Define the window for clipping (Window(row_start, col_start, height, w
        window = Window(*window_bounds)

        # Read the data within the window
        clipped_data = src.read(window=window)

        # Update the metadata for the clipped window
        out_meta = src.meta.copy()
        out_meta.update({
            "height": window.height,
            "width": window.width,
```

```
            "transform": src.window_transform(window)
        })

        # Save the clipped raster to the output file
        with rasterio.open(output_raster, 'w', **out_meta) as dest:
            dest.write(clipped_data)

    print(f"Clipped raster saved as: {output_raster}")

    # Normalize the data for visualization (assuming we have 3 bands for RGB)
    if clipped_data.shape[0] == 3:
        clipped_data_normalized = clipped_data.astype(np.float32)
        for i in range(3):
            min_val, max_val = clipped_data_normalized[i].min(), clipped_data_no
            clipped_data_normalized[i] = (clipped_data_normalized[i] - min_val)

        # Display the clipped raster
        plt.figure(figsize=(10, 10))
        plt.imshow(clipped_data_normalized.transpose(1, 2, 0))  # Transpose to g
        plt.title('Clipped Raster')
        # plt.axis('off')
        plt.show()
    else:
        print(f"The clipped data has {clipped_data.shape[0]} bands, not suitable

input_raster = r"C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\Lab-4
output_raster = r"C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\Lab-

# Define the window bounds (row_start, col_start, height, width) for clipping
# For example: (row_start=500, col_start=500, height=1000, width=1000)
window_bounds = (8000, 8000, 1000, 1000)

clip_and_save_raster(input_raster, output_raster, window_bounds)
```

```
Clipped raster saved as: C:\Users\Ranjeet Gupta\Downloads\Scientific Computing La
b\Lab-4\clipped_raster.tif
```



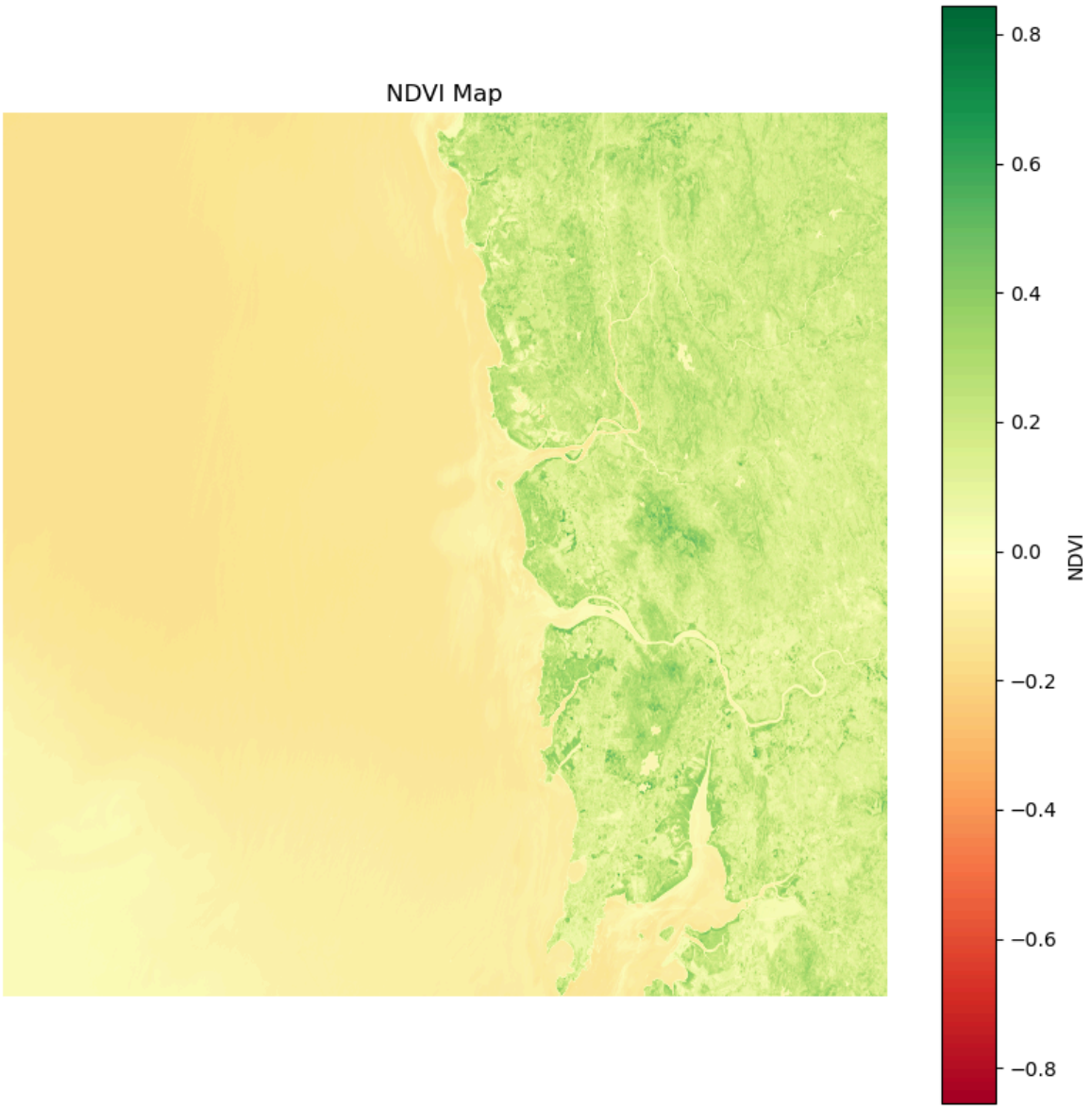
8) Create a NDVI map of your study area and display the results. Derive basic statistics from your NDVI file.

In [67]:

```
import rasterio
import numpy as np
import matplotlib.pyplot as plt

def calculate_ndvi(red_band_file, nir_band_file, output_ndvi_file):
    """
    Function to calculate NDVI and save the NDVI map.
    Returns:
    - None: The function saves the NDVI map and displays it.
    """

    # Open Red and NIR band files
    with rasterio.open(red_band_file) as red_src:
        red = red_src.read(1).astype('float32')  # Read red band
        red_meta = red_src.meta  # Get metadata for later use
```

```
    with rasterio.open(nir_band_file) as nir_src:
        nir = nir_src.read(1).astype('float32')  # Read NIR band

    ndvi = (nir - red) / (nir + red )

    # Update metadata for the output NDVI file
    red_meta.update(dtype=rasterio.float32, count=1, driver='GTiff') # Set drive

    # Save NDVI to a new file
    with rasterio.open(output_ndvi_file, 'w', **red_meta) as dst:
        dst.write(ndvi, 1)

    print(f"NDVI map saved as: {output_ndvi_file}")

    # Display NDVI map
    plt.figure(figsize=(10, 10))
    plt.imshow(ndvi, cmap='RdYlGn')
    plt.colorbar(label='NDVI')
    plt.title('NDVI Map')
    plt.axis('off')
    plt.show()

    # Calculate and print basic statistics
    ndvi_mean = np.mean(ndvi)
    ndvi_min = np.min(ndvi)
    ndvi_max = np.max(ndvi)
    ndvi_std = np.std(ndvi)

    print(f"NDVI Statistics:\nMean: {ndvi_mean}\nMin: {ndvi_min}\nMax: {ndvi_max

red_band_file = r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0
nir_band_file = r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0
output_ndvi_file = r"C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\L

calculate_ndvi(red_band_file, nir_band_file, output_ndvi_file)
```

```
NDVI map saved as: C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\Lab-
4\NDVI_output.tif
```




```
NDVI Statistics:
Mean: 0.00045799705549143255
Min: -0.8547894954681396
Max: 0.844084620475769
Standard Deviation: 0.15670810639858246
```

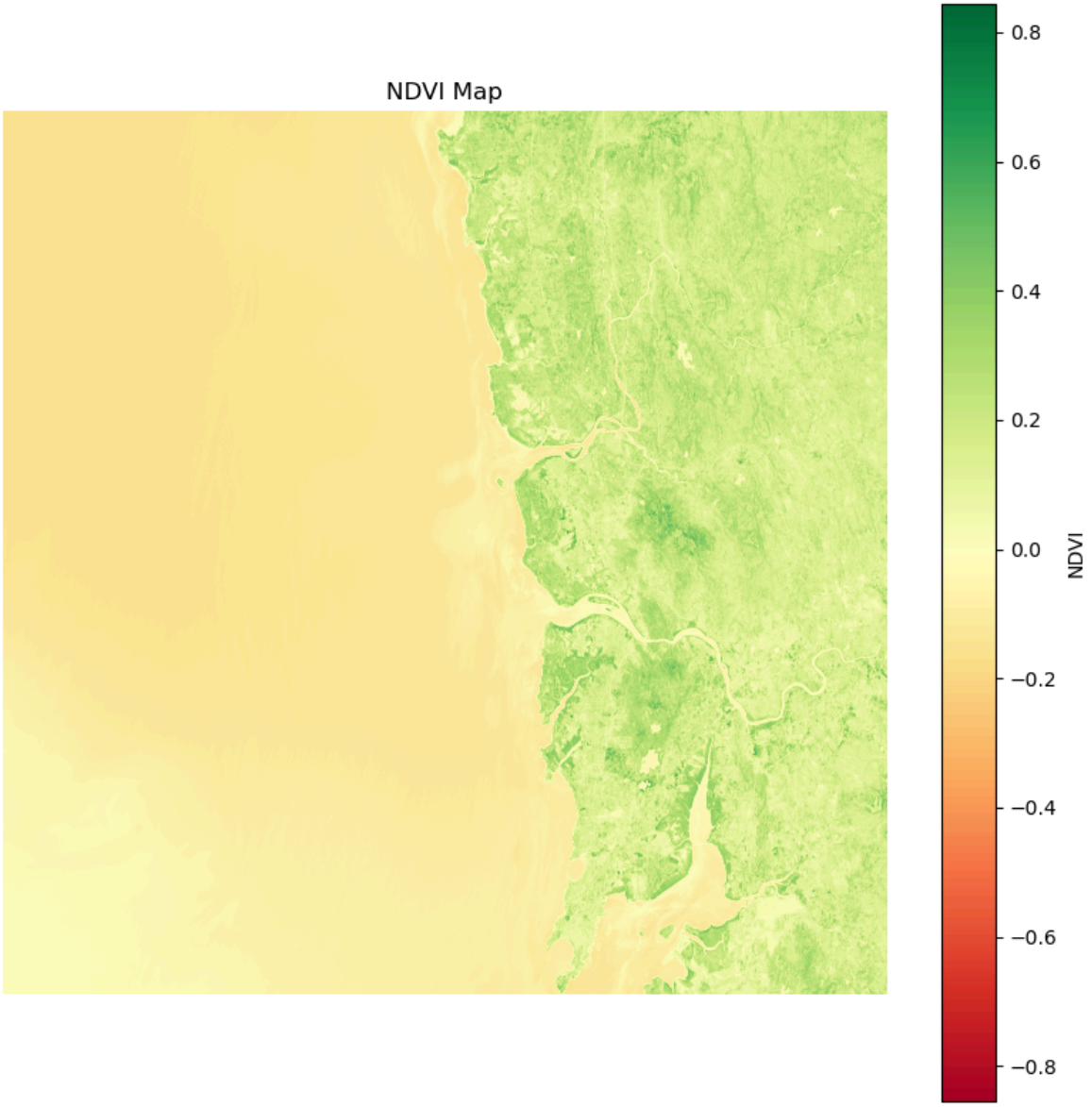
9) Create a function which will automate generation of NDVI map.

In [68]:

```python
import rasterio
import numpy as np
import matplotlib.pyplot as plt
import os

def generate_ndvi_map(red_band_file, nir_band_file, output_directory):
    """
    Automates the process of generating an NDVI map for given Red and NIR bands.

    Args:
    - red_band_file (str): File path for the red band.
    - nir_band_file (str): File path for the NIR band.
    - output_directory (str): Directory where the NDVI output will be saved.
```

```
    Returns:
    - output_ndvi_file (str): Path to the saved NDVI file.
    """

    # Check if the output directory exists, if not, create it
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)

    # Set output NDVI file path
    output_ndvi_file = os.path.join(output_directory, 'NDVI_output.tif')

    # Open Red and NIR band files
    with rasterio.open(red_band_file) as red_src:
        red = red_src.read(1).astype('float32')  # Read red band
        red_meta = red_src.meta  # Get metadata for later use

    with rasterio.open(nir_band_file) as nir_src:
        nir = nir_src.read(1).astype('float32')  # Read NIR band

    ndvi = (nir - red) / (nir + red)

    # Update metadata for the output NDVI file
    red_meta.update(dtype=rasterio.float32, count=1, driver='GTiff')

    # Save NDVI to a new file
    with rasterio.open(output_ndvi_file, 'w', **red_meta) as dst:
        dst.write(ndvi, 1)

    print(f"NDVI map saved as: {output_ndvi_file}")

    # Display NDVI map
    plt.figure(figsize=(10, 10))
    plt.imshow(ndvi, cmap='RdYlGn')
    plt.colorbar(label='NDVI')
    plt.title('NDVI Map')
    plt.axis('off')
    plt.show()

    # Calculate and print basic statistics
    ndvi_mean = np.mean(ndvi)
    ndvi_min = np.min(ndvi)
    ndvi_max = np.max(ndvi)
    ndvi_std = np.std(ndvi)

    print(f"NDVI Statistics:\nMean: {ndvi_mean}\nMin: {ndvi_min}\nMax: {ndvi_max

    return output_ndvi_file

# Example usage:
red_band_file = r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0
nir_band_file = r"C:\Users\Ranjeet Gupta\Downloads\S2B_MSIL2A_20240312T053639_N0
output_directory = r"C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\L

# Call the function to automate NDVI map generation
generate_ndvi_map(red_band_file, nir_band_file, output_directory)
```

```
NDVI map saved as: C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\Lab-
4\NDVI_output.tif
```




```
NDVI Statistics:
Mean: 0.00045799705549143255
Min: -0.8547894954681396
Max: 0.844084620475769
Standard Deviation: 0.15670810639858246
```

Out[68]: 'C:\\Users\\Ranjeet Gupta\\Downloads\\Scientific Computing Lab\\Lab-4\\NDVI_output.tif'

In [ ]: