Double-click (or enter) to edit

```
!pip install rasterio
```

```
Collecting rasterio
  Downloading rasterio-1.4.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.1 kB)
Collecting affine (from rasterio)
  Downloading affine-2.4.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: attrs in /usr/local/lib/python3.12/dist-packages (from rasterio) (25.4.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from rasterio) (2025.10.5)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.12/dist-packages (from rasterio) (8.3.0)
Collecting cligj>=0.5 (from rasterio)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.12/dist-packages (from rasterio) (2.0.2)
Collecting click-plugins (from rasterio)
  Downloading click_plugins-1.1.1.2-py2.py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.12/dist-packages (from rasterio) (3.2.5)
Downloading rasterio-1.4.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (22.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 22.3/22.3 MB 77.1 MB/s eta 0:00:00
Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Downloading click_plugins-1.1.1.2-py2.py3-none-any.whl (11 kB)
Installing collected packages: cligj, click-plugins, affine, rasterio
Successfully installed affine-2.4.0 click-plugins-1.1.1.2 cligj-0.7.2 rasterio-1.4.3
```

```
# ========== IMPORTS ==========
import numpy as np                              # numerical array operations (fast math on images)
import rasterio                                  # read/write GeoTIFFs and preserve geospatial metadata
from rasterio.enums import Resampling            # if we need to resample (not used here but handy)
from pathlib import Path                         # convenient and cross-platform path handling
import matplotlib.pyplot as plt                  # plotting (visualize NDVI quickly)
import warnings                                  # to issue warnings without crashing
```

```
from google.colab import files
# STEP 1: Upload your TIFF files into Google Colab
print("👉 Upload your 4 tif files (IMG_0455_1.tif to IMG_0455_4.tif)")
uploaded = files.upload()

# This will open a window where you can select files from your computer.
```

```
👉 Upload your 4 tif files (IMG_0455_1.tif to IMG_0455_4.tif)
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving IMG_0455_1.tif to IMG_0455_1 (1).tif
Saving IMG_0455_2.tif to IMG_0455_2 (1).tif
Saving IMG_0455_3.tif to IMG_0455_3 (1).tif
Saving IMG_0455_4.tif to IMG_0455_4 (1).tif
```

```
# STEP 2: List all files in the /content/ folder to confirm upload

import os

files_in_content = os.listdir('/content')   # Shows all files in the main Colab directory
print("Files in content:", files_in_content)
```

```
Files in content: ['.config', 'IMG_0455_1 (1).tif', 'IMG_0455_2.tif', 'IMG_0455_3 (1).tif', 'IMG_0455_4 (1).tif', 'IMG_0455_
```

```
# STEP 3: Set the folder path (always /content in Colab)
from pathlib import Path

FOLDER = Path("/content")   # All uploaded files go here

# Define the file names (same as uploaded)
BLUE_FILE  = FOLDER / "IMG_0455_1.tif"   # Blue band (480 nm)
GREEN_FILE = FOLDER / "IMG_0455_2.tif"   # Green band (550 nm)
RED_FILE   = FOLDER / "IMG_0455_3.tif"   # Red band (670 nm)
NIR_FILE   = FOLDER / "IMG_0455_4.tif"   # NIR band (850 nm)

# Check if all files exist (should print: True True True True)
print("Files exist:",
      BLUE_FILE.exists(),
      GREEN_FILE.exists(),
      RED_FILE.exists(),
      NIR_FILE.exists())
```

```
Files exist: True True True True
```

```
# STEP 4: Function to read a TIFF file

import rasterio

def read_band(file_path):
    """
    Reads a single-band TIFF file and returns:
    1) band = the pixel values as a NumPy array
    2) meta = metadata (required for saving output)
    """
    with rasterio.open(file_path) as src:
        band = src.read(1)          # Read first band (MicaSense TIFF has only one band)
        meta = src.meta.copy()      # Copy metadata: width, height, dtype, transform, CRS
    return band, meta
```

```
# STEP 5: Read the four TIFF files

blue,  meta_blue  = read_band(BLUE_FILE)    # Blue band
green, meta_green = read_band(GREEN_FILE)   # Green band
red,   meta_red   = read_band(RED_FILE)     # Red band
nir,   meta_nir   = read_band(NIR_FILE)     # Near-Infrared band

# Print shapes — all must match for NDVI calculation
print("Band shapes:")
print("Blue :", blue.shape)
print("Green:", green.shape)
print("Red  :", red.shape)
print("NIR  :", nir.shape)
```

```
Band shapes:
Blue : (968, 1280)
Green: (968, 1280)
Red  : (968, 1280)
NIR  : (968, 1280)
```

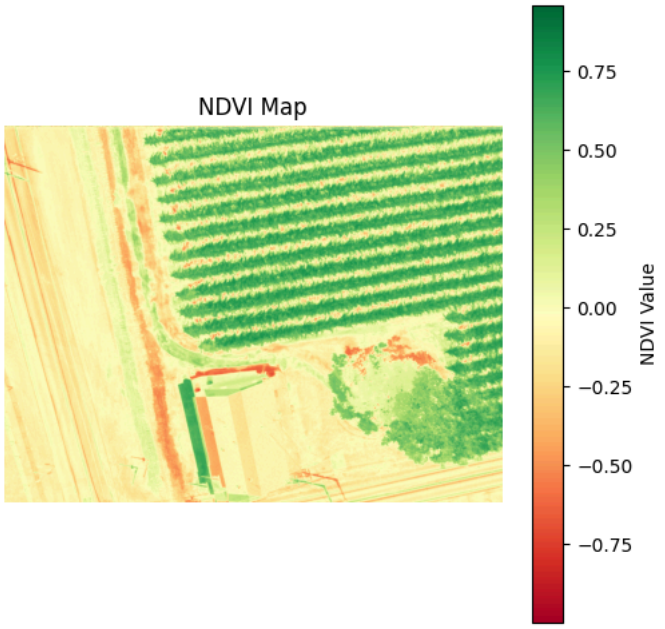
```
# STEP 6: Normalize DN (digital number) values to 0–1 scale
# This prevents NDVI from becoming huge numbers.

import numpy as np

blue_f  = blue  / np.max(blue)      # Convert blue band to reflectance-like values
green_f = green / np.max(green)     # Convert green band
red_f   = red   / np.max(red)       # Convert red band
nir_f   = nir   / np.max(nir)       # Convert NIR band

print("Reflectance normalization finished!")
```

```
Reflectance normalization finished!
```

```
# STEP 7: NDVI formula
# NDVI = (NIR - RED) / (NIR + RED)
# EPS prevents dividing by zero.

EPS = 1e-8

ndvi = (nir_f - red_f) / (nir_f + red_f + EPS)

# Print NDVI range to confirm values are between -1 and +1
print("NDVI value range:", ndvi.min(), "to", ndvi.max())
```

```
NDVI value range: -0.9999999800048831 to 0.9591836567693152
```

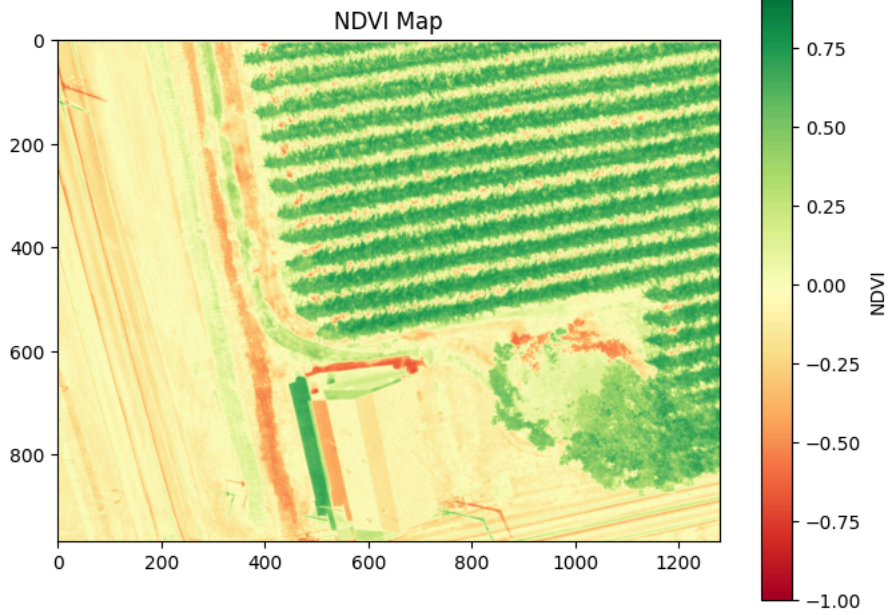
```
# STEP 8: Display NDVI as a color image

import matplotlib.pyplot as plt

plt.figure(figsize=(6,6))           # Set display size
plt.imshow(ndvi, cmap='RdYlGn')     # Vegetation colormap (green = healthy)
plt.colorbar(label="NDVI Value")    # Color scale on the side
plt.title("NDVI Map")
plt.axis("off")                     # Hide x/y axis
plt.show()
```




```
# STEP 9: Save NDVI in GeoTIFF format using metadata from the red band

meta = meta_red.copy()                          # Copy metadata
meta.update(dtype=rasterio.float32, count=1)   # Update data type and band count

output_path = "/content/NDVI_0455.tif"

# Write NDVI file
with rasterio.open(output_path, "w", **meta) as dst:
    dst.write(ndvi.astype(rasterio.float32), 1)

print("Saved NDVI file as:", output_path)
```

```
Saved NDVI file as: /content/NDVI_0455.tif
/usr/local/lib/python3.12/dist-packages/rasterio/__init__.py:366: NotGeoreferencedWarning: The given matrix is equal to Affi
  dataset = writer(
```

```
import rasterio
import matplotlib.pyplot as plt

# Open the NDVI file
with rasterio.open("NDVI_0455.tif") as src:
    ndvi = src.read(1)

plt.figure(figsize=(8,6))
plt.imshow(ndvi, cmap="RdYlGn")
plt.colorbar(label="NDVI")
plt.title("NDVI Map")
plt.show()
```



The NDVI map generated from the multispectral RedEdge dataset shows clear spatial variation in vegetation vigor. Higher NDVI values indicate areas with dense, healthy vegetation, while lower NDVI values represent sparse or stressed vegetation. The map effectively visualizes differences in plant canopy reflectance, allowing preliminary assessment of crop condition even from a single image.

This NDVI output serves as a basic introduction to vegetation index analysis, demonstrating how multispectral bands (Red and NIR) can