

Lec6



explain the code line by line and if there is error in the execution of the code then remove it by updating the code snippet and state the reason for the update



Please provide the specific code snippet you would like me to explain line by line and troubleshoot. This will allow me to offer a detailed explanation and suggest any necessary updates for error-free execution.



```
!pip install rasterio
```

```
raster_path = r'/content/drive/MyDrive/dl/s2_imagery/Hetauda_S2_2020_12.tif'
```

Remove SWIR1 & SWIR2 bands

```
import rasterio as rio
import numpy as np
```

```
src = rio.open(raster_path)
ds = src.read()
src.meta
```

```
{'driver': 'GTiff',
 'dtype': 'float32',
 'nodata': nan,
 'width': 6038,
 'height': 4103,
 'count': 6,
 'crs': CRS.from_epsg(4326),
 'transform': Affine(8.983152841195215e-05, 0.0, 84.68328351866317,
 0.0, -8.983152841195215e-05, 27.653108885636467)}
```

```
ds.shape
```

```
(6, 4103, 6038)
```

```
ds = ds.T
```

```

ds.shape

(6038, 4103, 6)

new_raster = ds[:, :, :4]

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,2, figsize=(12,5))
ax[0].imshow(new_raster[:, :, :3])
ax[0].set_title('RGB')
ax[1].imshow(new_raster[:, :, 3])
ax[1].set_title('NIR')

Text(0.5, 1.0, 'NIR')
/usr/local/lib/python3.10/dist-packages/matplotlib/cm.py:478: RuntimeWarning: invalid value
encountered in cast
xx = (xx * 255).astype(np.uint8)

out_meta = src.meta
out_meta.update({"count": 4})

new_raster = new_raster.T

with rio.open('/content/new_raster_4bands.tif', 'w', **out_meta) as dst:
    dst.write(new_raster)

Adding NDVI as new band

import rasterio as rio

src = rio.open(raster_path)
ds = src.read()
ds.shape

(6, 4103, 6038)

out_meta = src.meta
out_meta.update({'count': 7})

src.profile

{'driver': 'GTiff', 'dtype': 'float32', 'nodata': nan, 'width': 6038, 'height': 4103, 'count': 6, 'crs':
CRS.from_epsg(4326), 'transform': Affine(8.983152841195215e-05, 0.0, 84.68328351866317,
0.0, -8.983152841195215e-05, 27.653108885636467), 'blockxsize': 256, 'blockysize': 256, 'tiled':
True, 'compress': 'deflate', 'interleave': 'band'}

ds = ds.T

```

```

ds.shape

(6038, 4103, 6)

ndvi = (ds[:, :, 3] - ds[:, :, 0]) / (ds[:, :, 3] + ds[:, :, 0])
ndvi.shape

(6038, 4103)

import numpy as np
new_raster = np.zeros((ds.shape[0], ds.shape[1], 7))
new_raster[:, :, :6] = ds
new_raster[:, :, 6] = ndvi
new_raster.shape

(6038, 4103, 7)

np.nanmax(new_raster[:, :, 6]), np.nanmin(new_raster[:, :, 6])

(0.998606264591217, -0.31934934854507446)

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,4, figsize=(12,5))
ax[0].imshow(new_raster[:, :, :3])
ax[0].set_title('RGB')
ax[1].imshow(new_raster[:, :, 3])
ax[1].set_title('NIR')
ax[2].imshow(new_raster[:, :, 4])
ax[2].set_title('SWIR')
ax[3].imshow((new_raster[:, :, 6] + 1) / 3.0)
ax[3].set_title('NDVI')

Text(0.5, 1.0, 'NDVI')
/usr/local/lib/python3.10/dist-packages/matplotlib/cm.py:478: RuntimeWarning: invalid value
encountered in cast
xx = (xx * 255).astype(np.uint8)

new_raster = new_raster.T
new_raster.shape

(7, 4103, 6038)

with rio.open('/content/new_raster.tif', 'w', **out_meta) as dst:
    dst.write(new_raster)

```



Let's walk through the code line by line, explaining each step, and address any potential errors or updates required for smooth execution.

Step-by-Step Explanation:

1. Installing `rasterio`:

```
python

!pip install rasterio
```

- This command installs the `rasterio` library, which is used for reading and writing geospatial raster data. The `!` is used to run shell commands inside a Jupyter notebook.

2. Defining the path to the raster file:

```
python

raster_path = r'/content/drive/MyDrive/dl/s2_imagery/Hetauda_S2_2020_12.tif'
```

- Here, `raster_path` stores the file path of the Sentinel-2 imagery file. The `r` before the string makes it a raw string, avoiding escape sequences.

3. Importing necessary libraries:

```
python

import rasterio as rio
import numpy as np
```

- `rasterio` is imported for working with raster data, and `numpy` (`np`) is imported for handling numerical operations.

4. Opening the raster file and reading its data:

```
python

src = rio.open(raster_path)
ds = src.read()
```

- The raster file at `raster_path` is opened, and all bands are read into `ds`, which is a NumPy array.

5. Checking the metadata of the raster file:

```
python

src.meta
```

- This returns the metadata of the raster file, providing information about the number of bands, width, height, and CRS (coordinate reference system).

6. Checking the shape of the dataset:

```
python

ds.shape
```

- This will return the shape of the array as `(6, 4103, 6038)`, indicating 6 bands, 4103 rows, and 6038 columns.

7. Transposing the dataset:

```
python

ds = ds.T
ds.shape
```

- The dataset is transposed so that the bands become the last dimension. After transposition, the shape becomes `(6038, 4103, 6)`.

8. Removing the SWIR bands:

```
python

new_raster = ds[:, :, :4]
```

- This keeps only the first four bands (RGB + NIR) by slicing the array. SWIR1 and SWIR2 are removed.

9. Displaying the RGB and NIR bands:

```
python

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2, figsize=(12, 5))
ax[0].imshow(new_raster[:, :, :3]) # Display RGB
ax[0].set_title('RGB')
ax[1].imshow(new_raster[:, :, 3]) # Display NIR
ax[1].set_title('NIR')
```

- This code uses `matplotlib` to display the RGB bands (first 3 channels) in one subplot and the NIR band (4th channel) in another subplot.

10. Updating metadata to reflect the change in the number of bands:

python

```
out_meta = src.meta
out_meta.update({"count": 4})
```

- Since we reduced the number of bands from 6 to 4, the metadata is updated to reflect that.

11. Transposing the array back for writing:

python

```
new_raster = new_raster.T
```

- The dataset is transposed back to its original shape for saving. After transposing, the shape becomes `(4, 4103, 6038)`.

12. Saving the new raster with 4 bands:

python

```
with rio.open('/content/new_raster_4bands.tif', 'w', **out_meta) as dst:
    dst.write(new_raster)
```

- The updated raster with 4 bands is saved to a new file `new_raster_4bands.tif`.

Adding NDVI as a New Band

1. Re-opening the raster file:

python

```
src = rio.open(raster_path)
ds = src.read()
ds.shape
```

- The original raster file is reopened, and its data is read into `ds`.

2. Updating metadata for 7 bands:

python

```
out_meta = src.meta
```

```
out_meta.update({'count': 7})
```

- Since we are adding an NDVI band, the number of bands is updated to 7.

3. Transposing the dataset:

```
python

ds = ds.T
ds.shape
```

- The dataset is transposed again, changing the shape to `(6038, 4103, 6)`.

4. Calculating the NDVI:

```
python

ndvi = (ds[:, :, 3] - ds[:, :, 0]) / (ds[:, :, 3] + ds[:, :, 0])
```

- NDVI is calculated as `(NIR - Red) / (NIR + Red)`, where `ds[:, :, 3]` is the NIR band and `ds[:, :, 0]` is the Red band.

5. Creating a new raster array with 7 bands:

```
python

new_raster = np.zeros((ds.shape[0], ds.shape[1], 7))
new_raster[:, :, :6] = ds
new_raster[:, :, 6] = ndvi
```

- A new array is created with 7 bands: the first 6 from the original dataset and the 7th as the NDVI.

6. Visualizing RGB, NIR, SWIR, and NDVI bands:

```
python

fig, ax = plt.subplots(1, 4, figsize=(12, 5))
ax[0].imshow(new_raster[:, :, :3])
ax[0].set_title('RGB')
ax[1].imshow(new_raster[:, :, 3])
ax[1].set_title('NIR')
ax[2].imshow(new_raster[:, :, 4])
ax[2].set_title('SWIR')
ax[3].imshow((new_raster[:, :, 6] + 1) / 3.0)
ax[3].set_title('NDVI')
```

- This code visualizes RGB, NIR, SWIR, and the calculated NDVI in four subplots.

7. Transposing the array back:

```
python
```

```
new_raster = new_raster.T
```

- The dataset is transposed back for saving. After transposing, the shape becomes `(7, 4103, 6038)`.

8. Saving the new raster with 7 bands:

```
python
```

```
with rio.open('/content/new_raster.tif', 'w', **out_meta) as dst:
    dst.write(new_raster)
```

Errors and Fixes:

1. Warning with `matplotlib`:

```
bash
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/cm.py:478: RuntimeWarning: invalid
value encountered in cast
  xx = (xx * 255).astype(np.uint8)
```

- This warning occurs because some of the values in the image may contain `NaN` or `Inf`. To avoid this, normalize the data or handle `NaN` values:

```
python
```

```
ax[3].imshow(np.nan_to_num((new_raster[:, :, 6] + 1) / 3.0))
```

2. Ensure `NaN` values are handled:

- Use `np.nan_to_num` when processing arrays to prevent any issues related to `NaN` values in raster operations.

With these updates, the code should work smoothly without errors.



i think you missed some of the code snippet



Thank you for pointing that out! Let's go through the complete code again, making sure all parts are covered, and fix any potential issues, along with an explanation for each step.

Full Code Breakdown

```
python
```

```
!pip install rasterio
```

- **Explanation:** Installs the `rasterio` library, which is used for reading and writing raster data.
- **Potential Issue:** Ensure that you are running this in a Jupyter environment or a notebook where shell commands can be executed.

```
python
```

```
raster_path = r'/content/drive/MyDrive/dl/s2_imagery/Hetauda_S2_2020_12.tif'
```

- **Explanation:** The file path for the raster data is stored in `raster_path`. The `r` before the string indicates a raw string, meaning that backslashes are not treated as escape characters.

```
python
```

```
import rasterio as rio
import numpy as np
```

- **Explanation:** The necessary libraries (`rasterio` and `numpy`) are imported.

Reading the Raster Data

```
python
```

```
src = rio.open(raster_path)
ds = src.read()
```

- **Explanation:** The `raster_path` is opened using `rasterio.open`, and the bands from the raster file are read into `ds` (a NumPy array).

Metadata and Shape of the Dataset

```
python
```

```
src.meta
```

- **Explanation:** Displays the metadata of the raster file, including details such as the driver, dtype, width, height, and the number of bands (6 in this case).

```
python

ds.shape
```

- **Explanation:** This will print the shape of the dataset as `(6, 4103, 6038)`, indicating 6 bands, 4103 rows, and 6038 columns.

Transposing the Dataset and Removing SWIR Bands

```
python

ds = ds.T
ds.shape
```

- **Explanation:** The dataset is transposed so that the bands become the last dimension. After transposition, the shape changes to `(6038, 4103, 6)`.

```
python

new_raster = ds[:, :, :4]
```

- **Explanation:** This keeps only the first 4 bands (RGB + NIR) by slicing the array, effectively removing the SWIR1 and SWIR2 bands.

Visualizing RGB and NIR Bands

```
python

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2, figsize=(12, 5))
ax[0].imshow(new_raster[:, :, :3])
ax[0].set_title('RGB')
ax[1].imshow(new_raster[:, :, 3])
ax[1].set_title('NIR')
```

- **Explanation:** This code uses `matplotlib` to display the RGB (first 3 bands) in one subplot and the NIR band (4th band) in another subplot.

Updating Metadata and Saving the New Raster

```
python

out_meta = src.meta
out_meta.update({"count": 4})
```

- **Explanation:** The metadata is updated to reflect the removal of SWIR bands, and the count is updated to 4 (since there are only 4 remaining bands).

python

```
new_raster = new_raster.T
```

- **Explanation:** The dataset is transposed back to its original shape before writing to a file, changing the shape to `(4, 4103, 6038)`.

python

```
with rio.open('/content/new_raster_4bands.tif', 'w', **out_meta) as dst:
    dst.write(new_raster)
```

- **Explanation:** The new raster with 4 bands (RGB + NIR) is saved as `new_raster_4bands.tif`.

Adding NDVI as a New Band

python

```
import rasterio as rio
src = rio.open(raster_path)
ds = src.read()
ds.shape
```

- **Explanation:** Reopens the original raster file and reads it into `ds`.

python

```
out_meta = src.meta
out_meta.update({'count': 7})
```

- **Explanation:** The metadata is updated to reflect the addition of a new NDVI band. The count is updated to 7.

Calculating NDVI

python

```
ds = ds.T
```

- **Explanation:** Transposes the dataset again, making the shape `(6038, 4103, 6)`.

python

```
ndvi = (ds[:, :, 3] - ds[:, :, 0]) / (ds[:, :, 3] + ds[:, :, 0])
```

- **Explanation:** NDVI is calculated using the formula $(\text{NIR} - \text{Red}) / (\text{NIR} + \text{Red})$, where `ds[:, :, 3]` is the NIR band and `ds[:, :, 0]` is the Red band.

Creating a New Raster with NDVI Band

python

```
new_raster = np.zeros((ds.shape[0], ds.shape[1], 7))
new_raster[:, :, :6] = ds
new_raster[:, :, 6] = ndvi
```

- **Explanation:** A new array `new_raster` is created with 7 bands. The first 6 bands are copied from the original dataset, and the 7th band is the NDVI.

Visualizing RGB, NIR, SWIR, and NDVI

python

```
fig, ax = plt.subplots(1, 4, figsize=(12, 5))
ax[0].imshow(new_raster[:, :, :3])
ax[0].set_title('RGB')
ax[1].imshow(new_raster[:, :, 3])
ax[1].set_title('NIR')
ax[2].imshow(new_raster[:, :, 4])
ax[2].set_title('SWIR')
ax[3].imshow((new_raster[:, :, 6] + 1) / 3.0)
ax[3].set_title('NDVI')
```

- **Explanation:** The figure displays 4 subplots showing RGB, NIR, SWIR, and NDVI bands. NDVI is normalized for display by scaling it between 0 and 1.

Finalizing the Raster and Saving

python

```
new_raster = new_raster.T
new_raster.shape
```

- **Explanation:** The array is transposed back for saving, making the shape `(7, 4103, 6038)`.

python

```
with rio.open('/content/new_raster.tif', 'w', **out_meta) as dst:
    dst.write(new_raster)
```

- **Explanation:** The new raster file with the 7 bands, including NDVI, is saved as `new_raster.tif`.

Addressing Errors and Updates

1. Matplotlib Warning:

bash

```
/usr/local/lib/python3.10/dist-packages/matplotlib/cm.py:478: RuntimeWarning: invalid
value encountered in cast
  xx = (xx * 255).astype(np.uint8)
```

- **Explanation:** This warning indicates that some of the values in the dataset might be invalid (e.g., `NaN` or `Inf`). To avoid this, you can normalize the data or handle `NaN` values:

python

```
ax[3].imshow(np.nan_to_num((new_raster[:, :, 6] + 1) / 3.0))
```

- **Update:** Use `np.nan_to_num` to convert `NaN` values to 0 or another default value.

2. Handling `NaN` in NDVI:

- **Potential Issue:** If there are any invalid values in the NDVI calculation, it might lead to incorrect visualizations or errors. To handle this:

python

```
ndvi = np.nan_to_num((ds[:, :, 3] - ds[:, :, 0]) / (ds[:, :, 3] + ds[:, :, 0]))
```

3. Ensure Correct Transpositions:

- **Potential Issue:** Transposing the array back and forth can cause shape mismatches if not handled carefully. Ensure that you transpose the array back correctly before saving it.
- **Update:** The transpositions before saving ensure the array is in the correct format (bands-first format for rasterio).

With these updates, the code should now work without warnings or errors. Let me know if you'd like further clarification on any part!



explain the concept



The code you are working with involves processing satellite imagery, specifically working with raster data, and manipulating the bands within a raster image using Python libraries like `rasterio` and `numpy`. Let's break down the key concepts involved in this process:

1. Raster Data

Raster data refers to data in a grid format where each cell (or pixel) represents a specific geographic area. Satellite imagery, such as from the Sentinel-2 satellite, is commonly stored as raster data. A raster image contains several **bands** that capture different parts of the electromagnetic spectrum. Each band provides information about different wavelengths, such as red, green, blue (RGB), near-infrared (NIR), and shortwave infrared (SWIR).

For example:

- **Red, Green, and Blue (RGB)** bands capture visible light.
- **NIR (Near-Infrared)** band captures wavelengths outside the visible spectrum and is useful for vegetation analysis.
- **SWIR (Shortwave Infrared)** bands are helpful for analyzing water content in plants and soil, among other applications.

2. Raster Bands and Their Usage

Each band in a raster dataset represents data for a specific wavelength range. For example, in Sentinel-2 data, you may have:

- Band 2 (Blue)
- Band 3 (Green)
- Band 4 (Red)
- Band 8 (NIR)
- Band 11 (SWIR1)
- Band 12 (SWIR2)

In this code, you remove the **SWIR1 and SWIR2 bands** and work only with the first four bands (RGB + NIR). This is because for certain analyses, the SWIR bands may not be necessary, while the RGB and NIR bands are often used for vegetation and land cover studies.

3. Transposing Arrays

In Python, data from `rasterio` is loaded as a NumPy array, where the data is structured in a **band-first format**:

```
arduino
(bands, height, width)
```

This means the first dimension of the array corresponds to the number of bands, while the other two dimensions represent the image's height and width.

To work with this data easily, it is common to **transpose** the array to move the band dimension to the end:

```
arduino

(height, width, bands)
```

This change allows easier manipulation of the image data for tasks like visualizing the image (as RGB images use the last dimension for channels).

4. NDVI (Normalized Difference Vegetation Index)

NDVI is an important vegetation index derived from satellite data, widely used for monitoring plant health, biomass, and growth. It is calculated using the NIR and Red bands:

$$NDVI = \frac{(NIR - Red)}{(NIR + Red)}$$

Where:

- NIR is the Near-Infrared band, which captures light that plants reflect heavily when they are healthy.
- Red is the Red band, which captures visible light that plants absorb.

The result of NDVI calculation is a value between -1 and +1:

- Values closer to +1 indicate healthy vegetation.
- Values near 0 indicate bare soil or non-vegetated surfaces.
- Values below 0 can indicate water, clouds, or snow.

5. Handling Metadata

Raster files contain important **metadata**, which provides information about the file's properties, such as:

- **CRS (Coordinate Reference System):** Defines how the raster is georeferenced to a map.
- **Transform:** Provides the pixel size and the geographic location of the top-left corner.
- **Count:** Number of bands in the raster.

When modifying a raster (such as adding or removing bands), you need to update the metadata to reflect these changes before saving the file.

6. Removing Bands

In some cases, you may not need all the bands for a specific analysis. For instance, if the focus is on visualizing the RGB and NIR bands or calculating NDVI, the SWIR1 and SWIR2 bands may not be required. In the code:

```
python
```

```
new_raster = ds[:, :, :4]
```

This slices the data to keep only the first 4 bands (RGB + NIR), discarding the last two bands (SWIR1 and SWIR2).

7. Saving a New Raster

Once you modify the raster data, you may want to save it as a new file. Using `rasterio`, you can write the modified data back into a new raster file. This involves:

- Updating the metadata (such as the number of bands).
- Writing the data to a new file with the correct shape and metadata.

8. Visualization

For visualizing satellite imagery:

- **RGB** images are created using the first three bands.
- **NIR** and **NDVI** bands are often displayed in grayscale or as false-color composites to highlight certain features (e.g., vegetation health).

```
python
```

```
ax[0].imshow(new_raster[:, :, :3]) # RGB visualization
ax[1].imshow(new_raster[:, :, 3]) # NIR visualization
ax[3].imshow(ndvi)                 # NDVI visualization
```

9. Error Handling (NaN Values)

When calculating indices like NDVI, some of the data may be invalid (e.g., divide-by-zero cases). These lead to **NaN (Not a Number)** values in the array, which can cause errors or warnings during visualization. To avoid this, it's common to use `numpy` functions like `np.nan_to_num` to replace `NaN` values with a default number (usually 0).

Summary:

- **Raster Data:** Multi-band grid-like data, where each band captures different information (e.g., visible light, infrared).
- **Transposing:** Moving the band dimension to work with height and width more easily.
- **NDVI Calculation:** A vegetation index used to assess plant health.
- **Metadata:** Critical information about raster files that must be updated when modifying the file.
- **Visualization:** Displaying bands like RGB, NIR, or derived products like NDVI using tools like `matplotlib`.
- **Handling NaNs:** Ensuring that NaN values are addressed to prevent errors during processing or visualization.

These concepts form the foundation of working with raster data and performing satellite imagery analysis, and they are widely used in geospatial data science for tasks like land cover classification, vegetation monitoring, and environmental change detection.



okay



Glad to hear that the explanation was helpful! If you have any more questions or need further clarification on any topic, feel free to ask. Good luck with your project!

ChatGPT can make mistakes. Check important info.