Rasterio Tutorial

David Gerstenfeld, Adrian Terech

November 18, 2024

1.0 Introduction

This tutorial will showcase tools from the Rasterio library for Python. To showcase the basic use cases of Rasterio we are doing a mock analysis for the City of Philadelphia Planning Department and Sustinability Office on heat island issues comapred to current land cover & tree canpoy rates across the city. Rasterio is a Python library designed for reading and writing geospatial raster data. It provides a high-level API to interact with raster datasets, particularly those stored in formats like GeoTIFF. Raster data typically represents satellite imagery, aerial photography, or any spatially continuous variable (e.g., elevation or temperature) as a grid of pixels or cells.

It also handles reading and writing new GeoTIFFs and associated geographic metadata. Rasterio integrates well with the Geospatial Data Abstraction Library (GDAL), allowing access to spatial reference systems, projections, and other geospatial metadata. It enables easy reading of specific windows or blocks of large raster datasets without loading the entire file into memory. The raster data can be loaded directly into NumPy arrays for efficient numerical operations. Rasterio allows reading and transforming coordinate systems, making it easy to project raster data into different spatial reference systems.

In this tutorial we will cover reprojection, masking by using polygons, reclassifying rasters, zonal statistics, color coding, and using matplotlib to prepare a final map for the output raster.

Key features of Rasterio include:

- Reading and Writing GeoTIFFs: It can handle a variety of raster data formats (GeoTIFF, JPEG2000, etc.) with geographic metadata.
- Geospatial Metadata Handling: Rasterio integrates well with the Geospatial Data Abstraction Library (GDAL), allowing access to spatial reference systems, projections, and other geospatial metadata.
- Data Access: It enables easy reading of specific windows or blocks of large raster datasets without loading the entire file into memory.
- NumPy Integration: The raster data can be loaded directly into NumPy arrays for efficient numerical operations.
- Coordinate Reference Systems: Rasterio allows reading and transforming coordinate systems, making it easy to project raster data into different spatial reference systems.

Our tutorial main analysis will use rasterio and geopandas for processing of the data, and numPy for doing statistical analysis, and then matplotlib for output of map *Script Sections are Out of Order for Easy Explanation and Exercise Purposes*

Datasets Used

Shapefile Used

"PHL_Census_Tracts_2021.shp"

Raster files used

NLCD_TreeCoverCanopy_PhiladelphiaRegion_2021.tif

NLCD_LandCover_PhiladelphiaRegion_2021.tif

Land_Surface_Temperature_Lansat_2021.tif

1.0.1 Rasterio Installation & Data Preparation

We recommend installing Rasterio using anaconda within the Pysal geospatial library. We recommend you you install , you might want to use the gus5031 env.) To install, open the Miniconda prompt, navigate to the proper environment, and use the following commands:

conda create -n gus5031 -c conda-forge pysal geopandas #Installs Pysal which include Rasterio and Geopandas
conda activate gus5031 #The environment our class is using for tutorials

1.1 Importing all neccessary libraries and modules and functions

```
import pysal
import os
import geopandas as gpd
import numpy as np
import rasterio
import fiona
import subprocess
import rasterio.mask
import matplotlib.pyplot as plt
from rasterio.warp import calculate_default_transform, reproject, Resampling
from rasterio.transform import from_origin
```

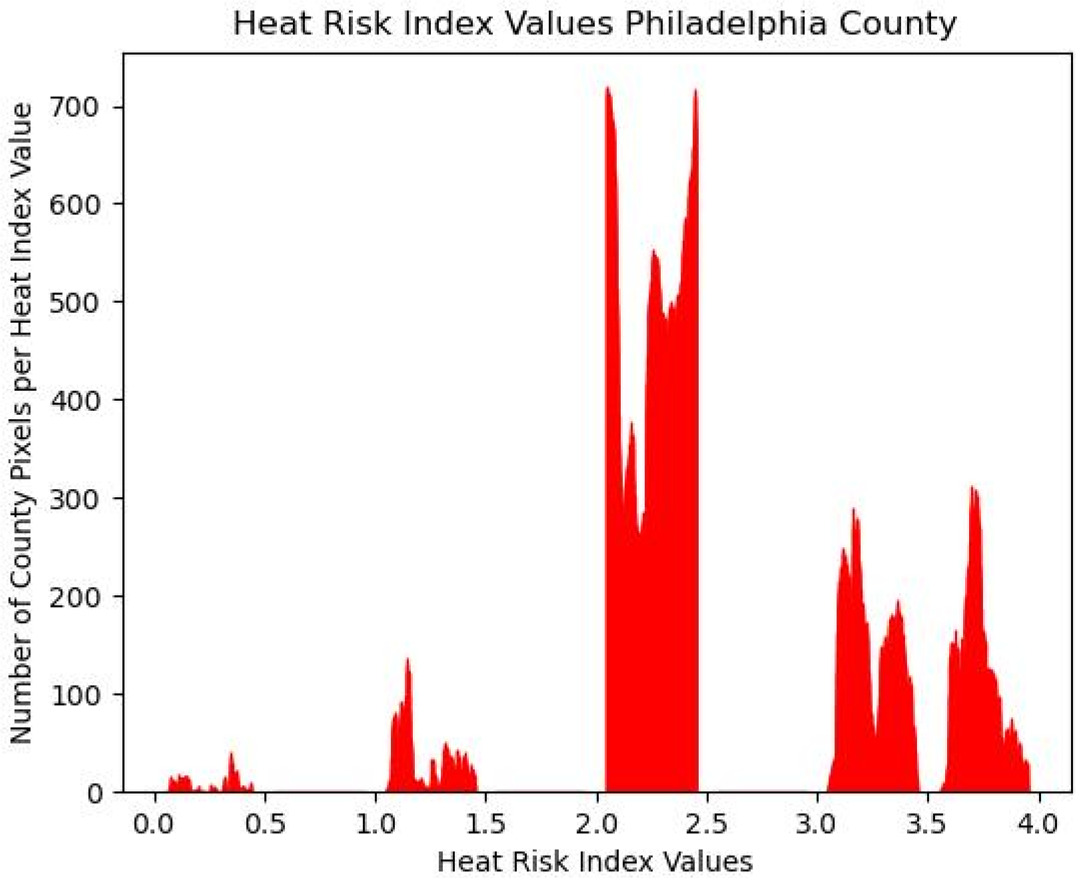
1.2 Setting Workspace and Labeling of Initial Variables

```
workspace = os.getcwd()

#planning_dist = "Planning_Districts.shp"
census_tracts = "PHL_Census_Tracts_2021.shp"
land_surf_temp = "Land_Surface_Temperature_Landsat_2021.tif"
land_cover = "NLCD_LandCover_PhiladelphiaRegion_2021.tif"
tree_cover = "NLCD_TreeCoverCanopy_PhiladelphiaRegion_2021.tif"
landsat_reprojected = 'LST_2021.tif'
landcover_reprojected = 'LC_2021.tif'
treecover_reprojected = 'TCC_2021.tif'
census_prj = 'census_nad_83.shp'
#planning_prj = 'planning_nad_83.shp'
dst_crs = 2272
```

```
#dst_crs equals the EPSG code for destionation reprojection which is NAD1983,
State Plane US PA South
```

2.0 [Actual Step #] Color and Scaling and Clipping data and Histogram to check data for null and outliers

```
# Define output path
output_path = 'heat_island_color.tif'

# Open input file
with rasterio.open(output_path_zonal) as src:
    data = src.read(1)
    meta = src.meta

# Define maximum value for scaling
max_value = 5.0

# Scale and clip data
clipped_data = np.clip(data, 0, max_value)
scaled_data = (clipped_data / max_value * 5).astype(np.uint8)

# Update metadata without nodata value
meta.update(dtype=rasterio.uint8)
if 'nodata' in meta:
    del meta['nodata']  # Remove nodata setting from metadata

# Save output file
with rasterio.open(output_path, 'w', **meta) as dst:
    dst.write(scaled_data, indexes=1)

#We used a histogram to help confirm that the code had no outliers or null data.
Below is the code for the histogram and the output.

plt.hist(scaled_data, bins=8, edgecolor='red')
plt.xlabel('Num_Of_Instances')
plt.ylabel('Heat_Index')
plt.title('Heat_Index_Philly')
plt.show()
```



As shown by the histogram output of the final processed data, all of the data is higher than 0 and less than 5 (however highest is actually less than 4). There are no null values shown or outliers. This helps confirm the accuracy of the data.

exercises:
easy
advanced

3.0 Reclassifying Rasters

3.1 Reclassifying Land Cover Raster

```
# Opening masked land cover raster
with rasterio.open("land_cover_mask.tif") as src:
raster_data = src.read(1)
profile = src.profile

# Create an empty array with the same shape as the raster data
reclassified_data = np.zeros_like(raster_data)

# Apply reclassification rules
reclassified_data[(raster_data > 24) | (raster_data < 21)] = 1
reclassified_data[(raster_data == 21)] = 2
reclassified_data[(raster_data == 22)] = 3
reclassified_data[(raster_data == 23)] = 4
```

```
reclassified_data[(raster_data == 24)] = 5

# Saving reclassified file
with rasterio.open('land_cover_mask_reclassified.tif', 'w', **profile) as dst:
  dst.write(reclassified_data, 1)
```

Land Cover was reclassified this way because values 21 to 24 indicate developed land, varying in development intensity (21 is the lowest intenity, 24 is the highest). Values of 1 to 5 were added to reclassified raster, with a high value indicating higher density and higher risk to urban heat island effect.

3.2 Reclassifying Tree Cover Raster

```
# Opening maked tree cover raster
with rasterio.open("tree_cover_mask.tif") as src:
  raster_data = src.read(1)
  profile = src.profile

# Create an empty array with the same shape as the raster data
reclassified_data = np.zeros_like(raster_data)

# Apply reclassification rules
reclassified_data[(raster_data >= 0) & (raster_data <= 20)] = 5
reclassified_data[(raster_data >= 21) & (raster_data <= 40)] = 4
reclassified_data[(raster_data >= 41) & (raster_data <= 60)] = 3
reclassified_data[(raster_data >= 61) & (raster_data <= 80)] = 2
reclassified_data[(raster_data >= 81) & (raster_data <= 100)] = 1

# Saving reclassified file
with rasterio.open('tree_cover_mask_reclassified.tif', 'w', **profile) as dst:
  dst.write(reclassified_data, 1)
```

Tree cover raster was split using the 5-class Jenks (Natural Breaks) method. Since lower tree cover increases risk to urban heat island effect, values were reclassified from 5 to 1.

3.3 Reclassifying Landsat Data Raster

```
# Opening masked landsat data raster
with rasterio.open("land_surf_temp_mask.tif") as src:
  raster_data = src.read(1)
  profile = src.profile

# Create an empty array with the same shape as the raster data
reclassified_data = np.zeros_like(raster_data)

# Applying reclassification rules
reclassified_data[(raster_data >= 50)] = 0
reclassified_data[(raster_data >= 51) & (raster_data <= 60)] = 1
```

```
reclassified_data[(raster_data >= 61) & (raster_data <= 70)] = 2
reclassified_data[(raster_data >= 71) & (raster_data <= 80)] = 3
reclassified_data[(raster_data >= 81) & (raster_data <= 90)] = 4
reclassified_data[(raster_data >= 91)] = 5

# Saving reclassified file
with rasterio.open('landsat_mask_reclassified.tif', 'w', **profile) as dst:
  dst.write(reclassified_data, 1)
```

Landsat data was reclassified into a 6-class method, where the highest and lowest class contain the outlier data while the interior 4 classes are split by 10 degrees. Higher temperature was given a higher reclassified value.

exercises:
easy
advanced

4.0 Reprojection of Census Vector Data

```
 gdf = gpd.read_file(census_tracts)

print("Original CRS:", gdf.crs)

gdf_reprojected = gdf.to_crs(dst_crs)

gdf_reprojected.to_file(census_reprojected, driver='ESRI Shapefile')

print("Reprojected CRS:", gdf_reprojected.crs)
```

exercises:
easy
advanced

5.0 Reprojection Loop for Raster Data

```
 source_rasters = tree_cover, land_surf_temp
    output_raster_paths = treecover_reprojected, landsat_reprojected
    output_first_raster_path = landcover_reprojected

    # Open the first raster and get its specifications
    with rasterio.open(land_cover) as target_raster:
      target_shape = (target_raster.height, target_raster.width)
      target_data = target_raster.read(1)
      source_dtype = target_data.dtype

    # Create an empty array for the reprojected target raster data
    destination_target = np.empty(target_shape, dtype=source_dtype)
```

```
    # Reproject the target raster
    with rasterio.open(land_cover_raster) as target_raster:
      target_data = target_raster.read(1)
      target_transform = target_raster.transform
      source_crs = target_raster.crs

    # Define the target CRS and resolution
    dst_transform, dst_width, dst_height =
rasterio.warp.calculate_default_transform(
        source_crs, dst_crs, target_raster.width, target_raster.height,
*target_raster.bounds)

    # Create a destination array for the reprojected target
    destination_target = np.empty((dst_height, dst_width), dtype=source_dtype)

    # Perform the reprojection
    reproject(
        source=target_data,
        destination=destination_target,
        src_transform=target_transform,
        src_crs=source_crs,
        dst_transform=dst_transform,
        dst_crs=dst_crs,
        resampling=Resampling.nearest
    )

    # Save the reprojected target raster
    with rasterio.open(
      landcover_reprojected,
      'w',
      driver='GTiff',
      height=dst_height,
      width=dst_width,
      count=1,
      dtype=source_dtype,
      crs=dst_crs,
      transform=dst_transform
    ) as dst:
      dst.write(destination_target, 1)

    print(f"Reprojected target raster saved as {landcover_reprojected}")

    # Looping through the remaining rasters with land cover as the target raster
    for source_raster_path, output_raster_path in zip(source_rasters,
output_raster_paths):
      with rasterio.open(source_raster_path) as source_raster:
        source_data = source_raster.read(1)
        source_transform = source_raster.transform
        source_crs = source_raster.crs
        source_dtype = source_data.dtype
```

```
        # Create an empty array with the shape and dtype of the target resolution
        destination = np.empty((dst_height, dst_width), dtype=source_dtype)

        # Perform the reprojection
        reproject(
            source=source_data,
            destination=destination,
            src_transform=source_transform,
            src_crs=source_crs,
            dst_transform=dst_transform,
            dst_crs=dst_crs,
            resampling=Resampling.nearest  # You can use other methods like
bilinear, cubic, etc.
        )

    # Save the reprojected raster to a new file
    with rasterio.open(
        output_raster_path,
        'w',
        driver='GTiff',
        height=dst_height,
        width=dst_width,
        count=1,
        dtype=source_dtype,
        crs=dst_crs,
        transform=dst_transform
    ) as dst:
        dst.write(destination, 1)

    print(f"Reprojected source raster saved as {output_raster_path}")
```

exercises:

easy

advanced

6.0 Masking Raster Data Using Polygons from Census Data

```
# File paths
input_files = [landcover_reprojected, treecover_reprojected, landsat_reprojected]
output_files = ["land_cover_mask.tif", "tree_cover_mask.tif",
"land_surface_temp_mask.tif"]

# Read the geometry shapes from shapefile
with fiona.open(census_reprojected, "r") as shapefile:
    shapes = [feature["geometry"] for feature in shapefile]

# Loop through each raster, apply mask, and save the output
for input_path, output_path in zip(input_files, output_files):
```

```
    with rasterio.open(input_path) as src:
        # Mask the raster with the shapefile geometries
        out_image, out_transform = rasterio.mask.mask(src, shapes, crop=True)
        out_meta = src.meta

        # Update metadata
        out_meta.update({
            "driver": "GTiff",
            "height": out_image.shape[1],
            "width": out_image.shape[2],
            "transform": out_transform
        })

        # Save the masked raster to the output file
        with rasterio.open(output_path, "w", **out_meta) as dest:
            dest.write(out_image)

    print(f'{input_path} has been masked and saved to {output_path}')
```

exercises:

easy

advanced

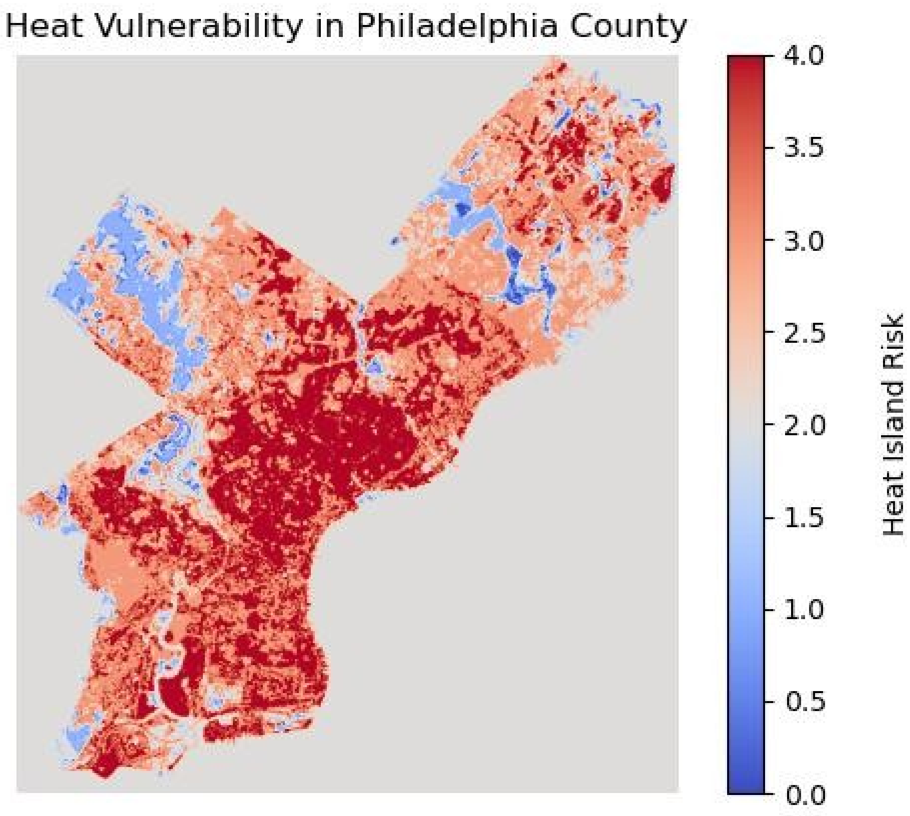
7.0 Zonal Statistics on Raster Outputs Using NumPy

```
# Defining input paths
raster_paths = ['land_cover_mask_reclassified.tif',
'tree_cover_mask_reclassified.tif', 'landsat_mask_reclassified.tif']
# Creating output file
output_path_zonal = 'heat_island_effect.tif'

# Opening input rasters
with rasterio.open(raster_paths[0]) as src:
  meta = src.meta  # Getting metadata from first raster
  # Reading and stacking all rasters
  stacked_data = np.stack([rasterio.open(path).read(1) for path in raster_paths])

# Calculating the average
average_data = np.nanmean(stacked_data, axis=0)

# Updating metadata
meta.update(dtype=rasterio.float32, count=1, nodata=np.nan)


with rasterio.open(output_path_zonal, 'w', **meta) as dst:
  dst.write(average_data, indexes=1)

print(f"Averaged raster saved as {output_path_zonal}")
```

exercises:
easy
advanced

8.0 Chloropleth Final Output

```
plt.imshow(scaled_data, cmap='coolwarm')
plt.axis('off')
cbar = plt.colorbar()
cbar.set_label('Heat Island Risk', labelpad=20)
plt.show()
```



exercises:
easy
advanced

Helpful Links for Resources on Rasterio

https://rasterio.readthedocs.io/en/stable/topics/index.html

https://geobgu.xyz/py/10-rasterio1.html#

Rasterio Exercises

```
Here are a few simple exercises to get familiar with using the rasterio library in
Python for working with raster data:

1. Open and Inspect a Raster File
Objective: Learn how to open a raster file and inspect its properties.
Instructions:
python
Copy code
import rasterio

Q1. Open a raster file and print the CRS, Width, Height, Bounds and Number of
bands in that file.
# Replace 'your_raster_file.tif' with the path to your raster file
with rasterio.open('your_raster_file.tif') as src:
    print("CRS:", src.crs)  # Coordinate Reference System
    print("Width:", src.width)
    print("Height:", src.height)
    print("Bounds:", src.bounds)
    print("Number of bands:", src.count)

2. Read and Display Raster Bands
Objective: Read a specific band and display it as an array.
Instructions:
python
Copy code
import matplotlib.pyplot as plt

# Read and plot the first band
with rasterio.open('your_raster_file.tif') as src:
    band1 = src.read(1)  # Reading the first band

plt.imshow(band1, cmap='gray')
plt.title("Band 1")
plt.colorbar()
plt.show()

3. Extract Raster Metadata
Objective: Extract and print metadata of the raster file.
Instructions:
python
Copy code
with rasterio.open('your_raster_file.tif') as src:
    metadata = src.meta
print(metadata)

4. Crop a Raster File by Bounding Box
Objective: Use a bounding box to crop the raster data.
Instructions:
python
Copy code
```

```
from rasterio.windows import Window

# Define a bounding box and crop
with rasterio.open('your_raster_file.tif') as src:
    # Define the window with start and end coordinates in pixels
    window = Window(100, 100, 200, 200)
    cropped_data = src.read(1, window=window)

plt.imshow(cropped_data, cmap='gray')
plt.title("Cropped Data")
plt.colorbar()
plt.show()

5. Calculate NDVI (Normalized Difference Vegetation Index)
Objective: If you have a multi-band raster (e.g., with Red and NIR bands),
calculate NDVI.
Instructions:
python
Copy code
with rasterio.open('multiband_raster_file.tif') as src:
    nir = src.read(4)  # NIR band
    red = src.read(3)  # Red band

# Calculate NDVI
ndvi = (nir - red) / (nir + red)

plt.imshow(ndvi, cmap='RdYlGn')
plt.title("NDVI")
plt.colorbar()
plt.show()

These exercises cover basic raster handling tasks with rasterio, giving you a
hands-on way to understand and manipulate raster data. Let me know if you'd like
any additional examples!
```

DATA SOURCE LINKS

Land Cover and Tree Canopy Cover: https://www.mrlc.gov/viewer/ Downloaded using custom extent

Landsat Data: https://earthexplorer.usgs.gov/ Downloaded Band 10 dataset and metadata file. Band 10 data came in as raw pixel data, which had to converted to radiance, then to Kelvin, and then to Fahrenheit

Census Tracts: https://www.census.gov/cgi-bin/geo/shapefiles/index.php?year=2021&layergroup=Census+Tracts

Planning Districts: https://opendataphilly.org/datasets/planning-districts/

EXERCISE DATA SOURCE LINKS

https://opendataphilly.org/datasets/digital-elevation-model-dem/

WORKS CITED

Amindin, Atiyeh, et al. “Spatial and Temporal Analysis of Urban Heat Island Using Landsat Satellite Images.” Environmental Science and Pollution Research, vol. 28, no. 30, 30 Mar. 2021, pp. 41439–41450, https://doi.org/10.1007/s11356-021-13693-0.

Atasoy, Murat. “Assessing the Impacts of Land-Use/Land-Cover Change on the Development of Urban Heat Island Effects.” Environment, Development and Sustainability, 29 Nov. 2019, https://doi.org/10.1007/s10668-019-00535-w.

“Documentation — GeoPandas 0.11.0+0.G1977b50.Dirty Documentation.” Geopandas.org, geopandas.org/en/stable/docs.html.

Dorman, Michael . “Rasters (Rasterio) — Spatial Data Programming with Python.” Geobgu.xyz, 2023, geobgu.xyz/py/10-rasterio1.html. Accessed 16 Nov. 2024.

Matplotlib. “Matplotlib: Python Plotting — Matplotlib 3.3.4 Documentation.” Matplotlib.org, matplotlib.org/stable/index.html.

NumPy. “Overview — NumPy V1.19 Manual.” Numpy.org, 2022, numpy.org/doc/stable/.

Shahfahad, et al. “Land Use/Land Cover Change and Its Impact on Surface Urban Heat Island and Urban Thermal Comfort in a Metropolitan City.” Urban Climate, vol. 41, Jan. 2022, p. 101052, https://doi.org/10.1016/j.uclim.2021.101052.