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 basics of Rasterio, we are doing a mock analysis for the City of Philadelphia Planning Department and Sustinability Office on Urban Heat Island Effect. This study will be done by comparing the current land cover & tree canopy 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.

The Rasterio library handles reading and writing GeoTIFFs and other associated forms of geographic metadata. It 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 also allows reading and transforming coordinate reference systems, making it easy to project raster data into different spatial reference systems.

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.

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. Our tutorial will use Rasterio and Geopandas to process the data, NumPy to conduct

statistical analysis, and then Matplotlib for data visualization.

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 that you install Rasterio using 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 Geopand
conda activate gus5031  #The environment our class is using for tutorials
```

1.1 Importing all neccessary libraries and specific functions

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

1.2 Setting Workspace and Labeling of Initial Variables

Below code sets current directory as the workspace using the os library **getcwd()** function. This is then stored in the workspace variable.

```
workspace = os.getcwd()
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'
```

Script Sections are Out of Order for Easy Explanation and Exercise Purposes

---

2.0 [Actual Step #7] Color coding, Scaling, Clipping data and Histogram to check data for null and outliers

The code below defines the variable output path for the Geotif file.

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

The code uses rasterio to open the *output_path_zonal* raster file as the local variable *src*. Then *src*'s first band is read and stored in the variable data. Then the meta data from *src* is stored in the variable *meta*.

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

5.0 is stored in the *max_value* variable as the upper float limit for scaling the raster values.

```
# Define maximum value for scaling
max_value = 5.0
```

The NumPy function **np.clip**, clips the raster values from the input variable data from 0 to the variable *max_value* (which is 5.0). Any values outside of the range are set to the closest range bound. This is all stored in the *clipped_data* variable. Then the *clipped_data* variable is divided by the *max_value* variable and multiplied by the value 5. Finally the resultant data is converted to unsigned 8-bit positive integers using the NumPy **astype()** function.

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

Then the metadata is updated again using the update function with the dtype as rasterio unsigned 8-bit integers. An if conditional statement is then used in the updated **meta function** to delete any 'nodata' data within the *meta* variable.

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

Using the **rasterio open** function the updated *meta* variable data is written into the *output_path* variable and the *scaled_data* pixel data is written into *output_path* by using the write function, with one index stated.

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

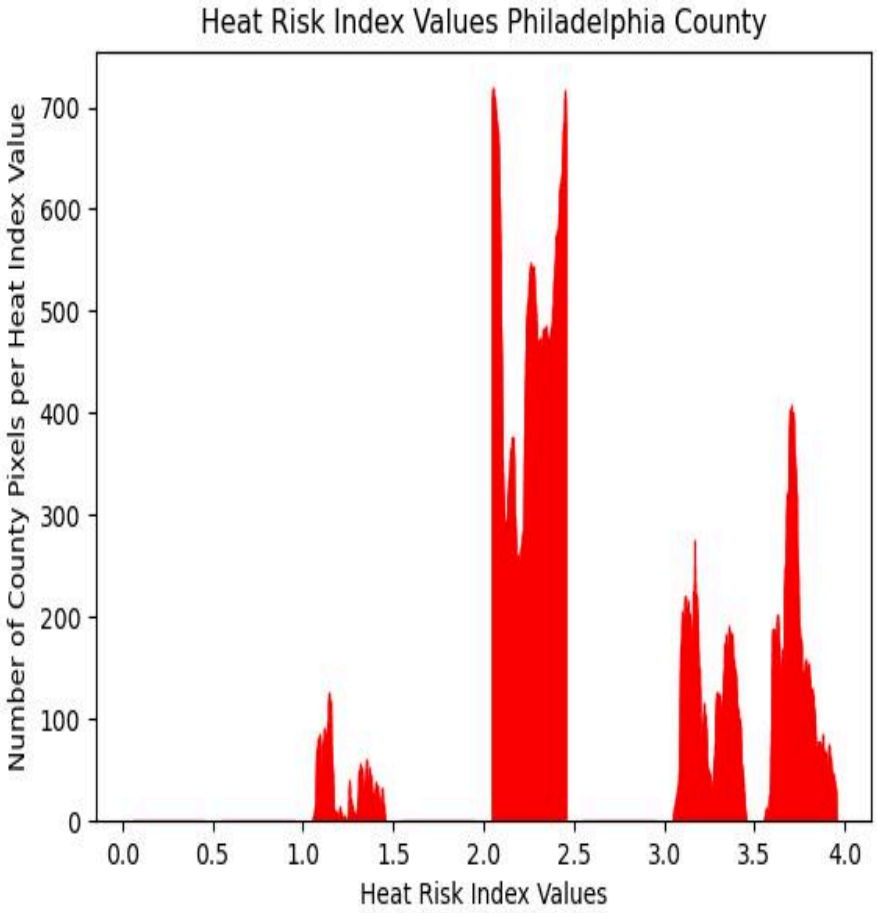
A histogram is made using matplotlib library initially with the **hist()** function. This is using the *scaled_data* input variable, a bins value of 8 for 8 histogram value categories, and edgecolor for color of the histogram shown.

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')
```

The **xlabel, ylabel, and title** functions result in the labeling for the histogram. The **show()** function displays the histogram in a visualization window.

```
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.

3.0 [Actual Step #5] Reclassifying Rasters

Look to "Reclassification" Workbook in the GitHub repository for this chapter.

4.0 [Actual Step #2] Reprojection of Census Vector Data

The below code uses the **gpd.read_file** function, to read the census_tracts shapefile. The resulting GeoDataFrame is stored in the *gdf* variable. Below that is a print statement which prints the input coordinate system using the command *gdf.crs*.

```
gdf = gpd.read_file(census_tracts)
print("Original CRS:", gdf.crs)
```

The code uses the **gdf.to_crs()** function with the argument input *dst_crs* variable to reproject the gdf variable into the *dst_crs*'s EPSG projection code.

```
gdf_reprojected = gdf.to_crs(dst_crs)
```

The **gdf_reprojected.to_file()** function writes the reprojected GeoDataFrame to a new file with an output file path labeled as the *census_reprojected* variable. The driver 'ESRI Shapefile' stated indicates the format of the output file which is a shapefile.

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

The print statement uses the *gdf_reprojected.crs* command to print the new coordinate reference system EPSG number.

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

5.0 [Actual Step #3] Reprojecting Raster Data

Look to "Reprojection" Workbook in the GitHub repository for this chapter.

6.0 [Actual Step #4] Masking Raster Data Using Polygons from Census Data

Look to "Masking Raster" Workbook in the GitHub repository for this chapter.

7.0 [Actual Step #6] Zonal Statistics on Raster Outputs Using NumPy

The script below conducts zonal statistics on three raster files. The first line of code creates a list storing the input raster files within variable *raster_paths*, while the second line of code stores the output file paths under the variable *output_path_zonal*.

```
# Defining input paths
raster_paths = ['land_cover_mask_reclassified.tif', 'tree_cover_mask_reclassified.tif', 'landsat_mask_reclassi
# 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])
```

Opens the first raster file in the list and stores it in the local *src* variable. The *src* variable meta data is then stored in the *meta* variable. List comprehension (using an iterating variable "path") is then used by the **rasterio open** function to read the first band of each raster in *raster_paths*. This function is used as an argument within the NumPy **np.stack** function to combine all the rasters into a single 3D NumPy array called *stacked_data*. The first dimension corresponds to the number of rasters. The second and third dimensions represent the spatial dimensions which are the rows and columns of the rasters.

```
# Calculating the Piexl-Wise average
average_data = np.nanmean(stacked_data, axis=0)
```

The NumPy nanmean function is then used with an input argument of *stacked_data* and uses the 0 axis which corresponds to the first dimension. This calculates the average for each pixel in the raster array while ignoring all nodata values. The result, *average_data*, is a 2D array representing the averaged raster.

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

This code updates the *meta* variable, while confirming the dtype data type is a float32 type raster file. The count sets the number of bands in output raster to 1. Finally, the nodata argument sets any no data value to “NaN”.

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

Rasterio open function is used on the *output_path_zonal* variable to write the update metadata to it, and write to it as *dst*, the *average_data* stored data with one index.

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

This print function outputs the message along with the variable file name for *output_path_zonal*.

8.0 [Actual Step #8] 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()
```

The below line of code uses the matplotlib library **imshow** function to display the input array *scaled_data* as an image. The second argument uses the color map “coolwarm” which represents a continuous range from blue to red which is suited for heat related data visualization.

```
plt.imshow(scaled_data, cmap='coolwarm')
```

The **axis(‘off’)** function removes the x and y axes tick marks as well as labels for a cleaner image.

```
plt.axis('off')
```

The **colorbar()** function adds a colorbar alongside the image to indicate the range of values and their corresponding colors, akin to a legend. It also sets the colorbar equal to the variable cbar.

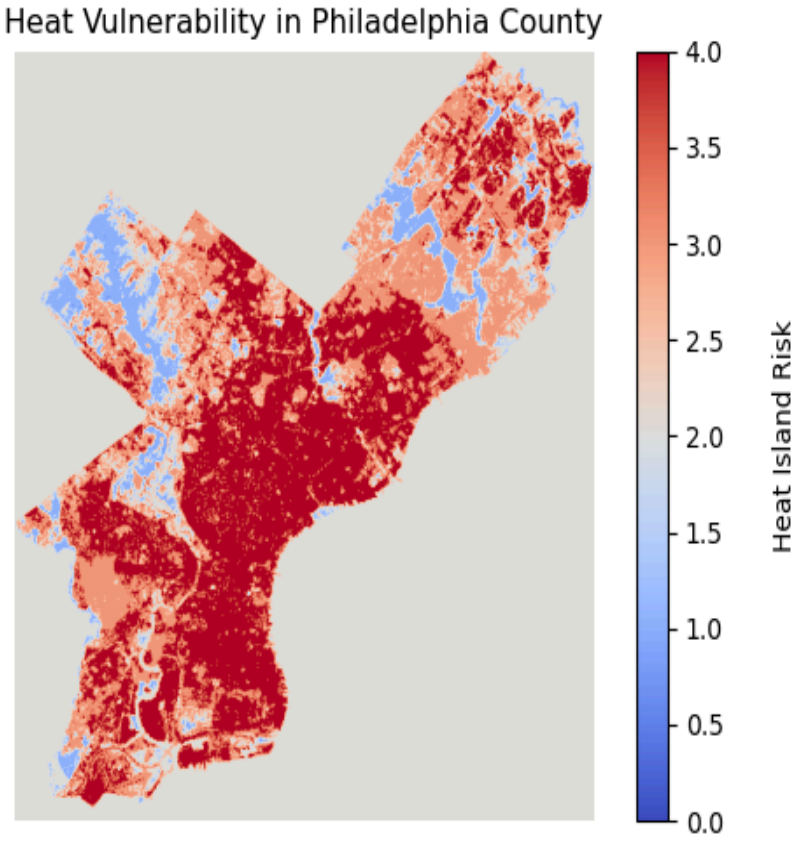
```
cbar = plt.colorbar()
```

The **set_label()** function adds a label to the color bar, and the labelpad argument adds that many units of padding between the color bar and the label.

```
cbar.set_label('Heat Island Risk', labelpad=20)
```

The **show()** function displays the image and color bar in a visualization window.

```
plt.show()
```



Helpful Links for Resources on Rasterio

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

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

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/

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.