3.0 [Actual Step #5] Reclassifying Rasters

By David Gerstenfeld, Adrian Terech

Introduction

In this chapter, we will go over the concept of reclassifying raster datasets and explaining their use for raster analysis. Some common instances where reclassification is used include simplifying datasets with a high number of unique values, reclassifying to only include values of interest, and assessing land cover change. Throughout this chapter, the reclassification script used in the final product will be split and explained step-by-step.

3.1 Preparing to reclassify

For this chapter, we will be observing the reclassification code used for the landsat dataset in the final result. The first section of code opens and acquires information from the specified raster dataset.

```
with rasterio.open("land_surf_temp_mask.tif") as lc:
    raster_data = lc.read(1)
    profile = lc.profile
```

The 'with' statement is used to open the raster dataset with the use of the **rasterio.open** function, where it is then defined as variable *lc*. The variable *lc* is used in the following two lines to read the first band and metadata of the raster dataset. Variable *raster_data* is used to store the first band within the *lc* dataset, while variable *profile* is used to store its profile metadata. Both variables will be used later in the script and thus, are important to define.

After the variables have been defined, a new NumPy array is created to convert all the existing raster pixel values to 0 using the **np.zeros_like** function as seen below:

```
reclassified_data = np.zeros_like(raster_data)
```

The defined variable *raster_data* from the previous section is added to the function to convert all the raster values within the land cover dataset to 0, giving the user a blank slate to reclassify them in a way that better suits the user.

3.2 Reclassification

The next step is to reclassify the dataset with new values. This is accomplished by utilizing a specified number of rules that set newly defined values based on where they fall under the set of rules established by the user.

Here's an example of where a set of rules are used to reclassify the landsat dataset:

```
reclassified_data[(raster_data <= 59.50)] = 1
reclassified_data[(raster_data >= 59.51) & (raster_data <= 69.50)] = 2
reclassified_data[(raster_data >= 69.51) & (raster_data <= 79.50)] = 3
reclassified_data[(raster_data >= 79.51) & (raster_data <= 89.50)] = 4
reclassified_data[(raster_data >= 89.51)] = 5
```

You can also use the **np.round** function to round pixel values before they're reclassified. This can be helpful for reclassifying raster datasets with decimal values. When using the command, it is recommended to include it after the first band of the raster dataset has been read, but also before the NumPy array is created. Here's an instance of where the **np.round** function is used:

```
with rasterio.open("land_surf_temp_mask.tif") as src:
    raster_data = src.read(1)
    rounded_data = np.round(raster_data)
    profile = src.profile

reclassified_data = np.zeros_like(raster_data)
```

In the reclassification section, the variable *raster_data* is substituted by *rounded_data*.

```
reclassified_data[(rounded_data < 60)] = 1
reclassified_data[(rounded_data >= 60) & (rounded_data < 70)] = 2
reclassified_data[(rounded_data >= 70) & (rounded_data < 80)] = 3
reclassified_data[(rounded_data >= 80) & (rounded_data < 90)] = 4
reclassified_data[(rounded_data >= 90)] = 5
```

You can also reclassify raster datasets from a numerical value to a string value in some cases: Example:

```
reclassified_data[(rounded_data < 60)] = ‘Very Low’
reclassified_data[(rounded_data >= 60) & (rounded_data < 70)] = ‘Low’
reclassified_data[(rounded_data >= 70) & (rounded_data < 80)] = ‘Moderate’
reclassified_data[(rounded_data >= 80) & (rounded_data < 90)] = ‘High’
reclassified_data[(rounded_data >= 90)] = ‘Very High’
```

Keep in mind that whenever you’re reclassifying a numeric value into a string value, or vice versa, the newly created array needs to specify the new data type used to store the reclassified data. For example:

Numeric to String

```
reclassified_data = np.char.array(np.empty_like(raster_data, dtype='<U20'))
```

Note that the **np.zeros_like** function is replaced by the **np.char.array** and **np.empty_like** functions. This is because the **np.zero_like** function is only capable of handling numeric values, while **np.char.array** and **np.empty_like** are capable of handling string values. The data type is also specified in the code to a 20-character limit Unicode array, which is capable of holding string values. The character limit of a Unicode array can be altered to better fit the user, for example ‘U50’ would establish a 50-character limit on the array.

When reclassifying from a string value to a numeric value, you only need to specify the data type that it is being converted to (either dtype=int or dtype=float). Also note that when you’re reclassifying from an integer type to a float value and vice versa, you need to specify the data type.

3.3 Saving reclassified rasters

The final step in the reclassification process is to save the reclassified data into a new output file. We’ll call it *landsat_reclass* to save the output file to the specified name earlier in the script. Using **rasterio.open**, the output file is opened in a writing mode (‘w’), while the metadata is added to ensure that the new output file uses the same metadata as the input file. In the final line, the reclassified data is written into the first band of the output file by using **dest.write**.

```
with rasterio.open(landsat_reclass, 'w', **profile) as dest:
    dest.write(reclassified_data, 1)
```

3.4 Reclassifications for Final Result

Listed below are the rulesets used to reclassify the land cover and tree cover datasets:

Land Cover

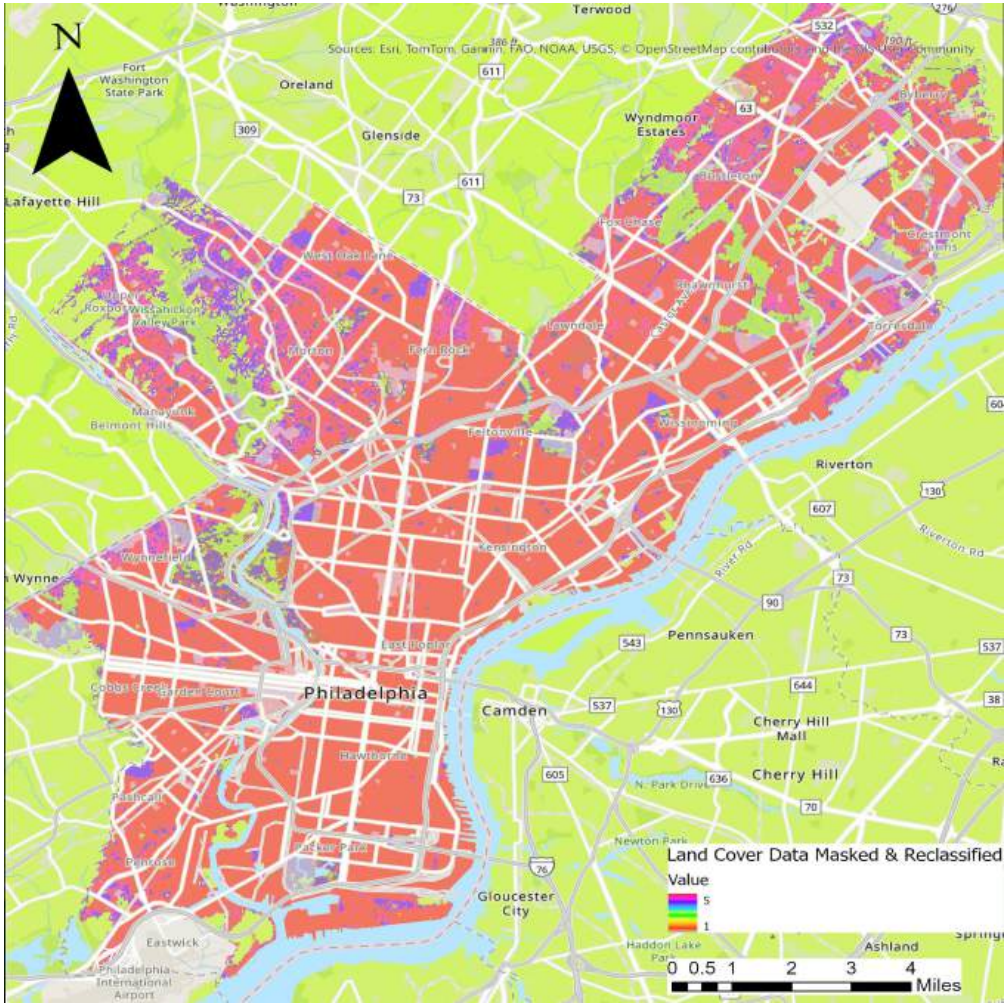
```
reclassified_data[(raster_data > 24) | (raster_data < 21)] = 1
reclassified_data[(raster_data == 21)] = 2
reclassified_data[(raster_data == 22)] = 3
reclassified_data[(raster_data == 23)] = 4
reclassified_data[(raster_data == 24)] = 5
```

Tree Cover

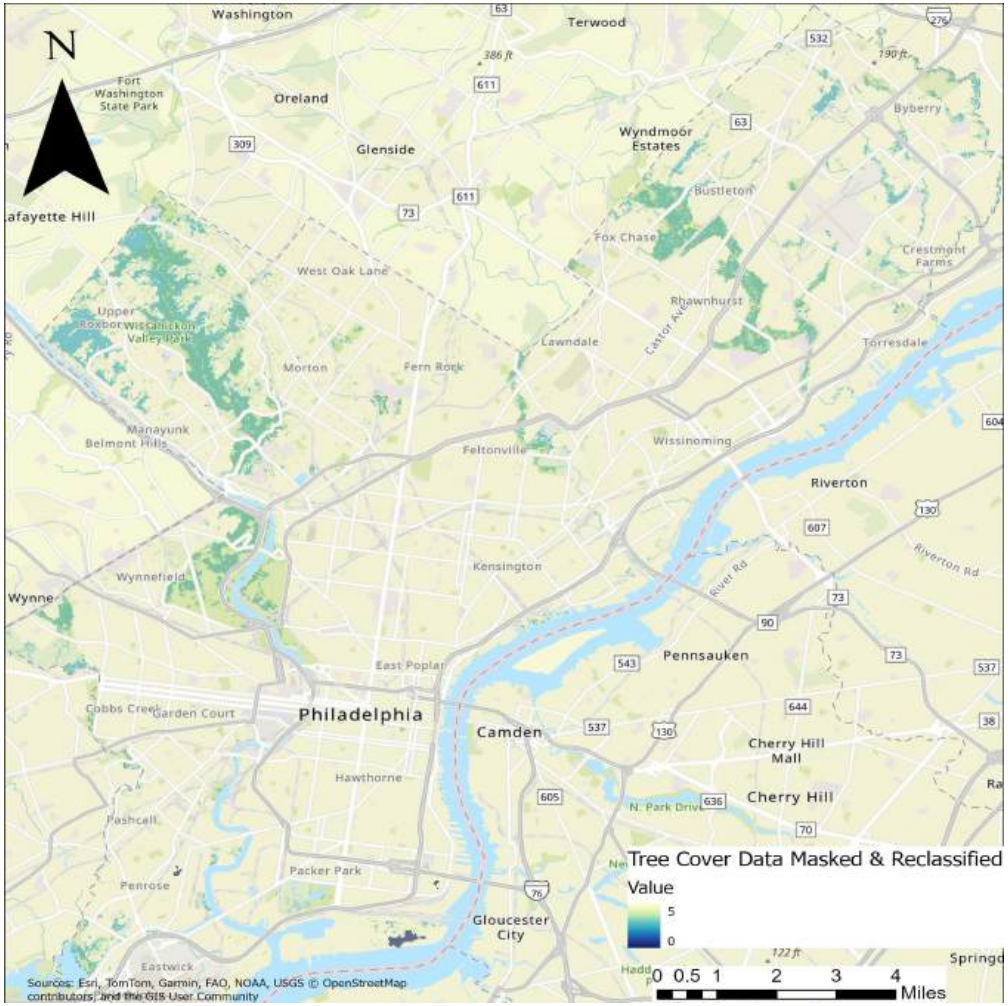
```
reclassified_data[(raster_data >= 0) & (raster_data <= 20)] = 5
reclassified_data[(raster_data >= 21) & (raster_data <= 40)] = 4
reclassified_data[(raster_data >= 41) & (raster_data <= 60)] = 3
reclassified_data[(raster_data >= 61) & (raster_data <= 80)] = 2
reclassified_data[(raster_data >= 81)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

Reasonings

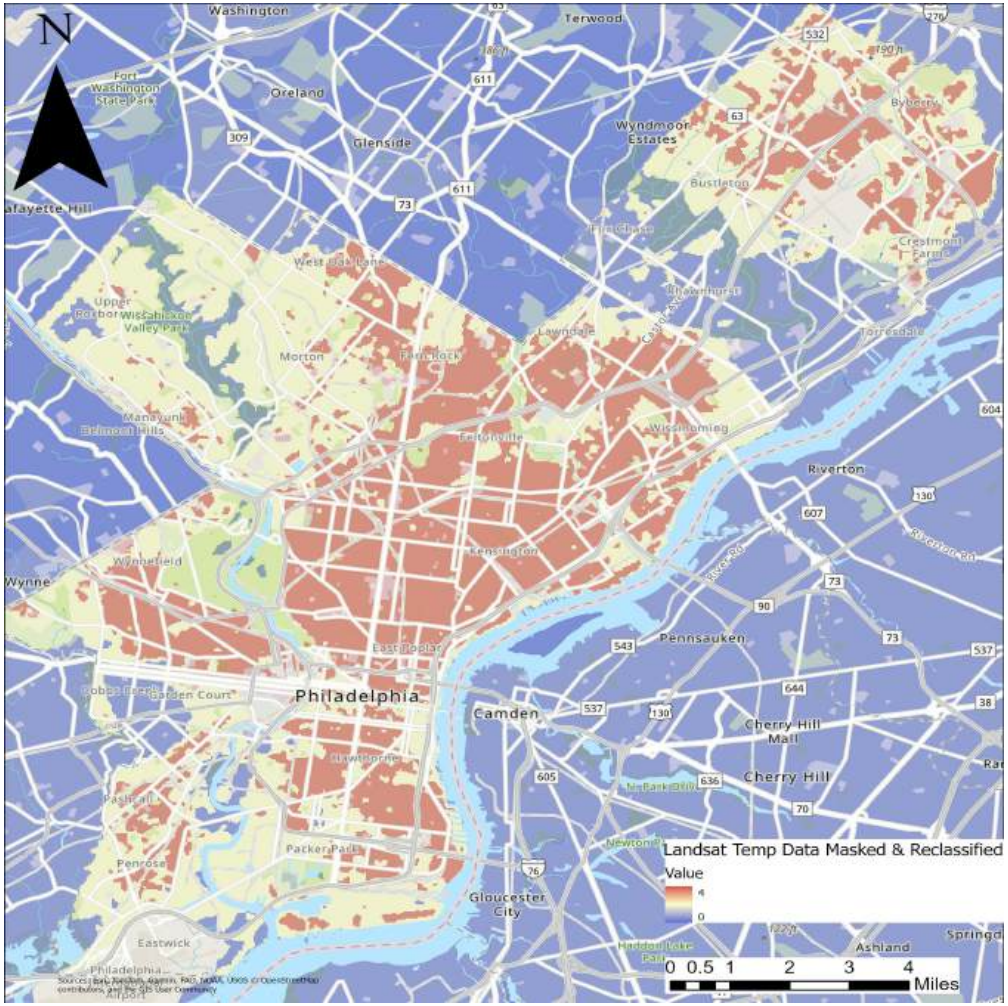
Land Cover was reclassified this way because values 21 to 24 indicate developed land, varying in development intensity (21 is the lowest intenity, 24 is the highest). Values of 1 to 5 were added to reclassified raster, with a high value indicating higher density and higher risk to urban heat island effect.



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



Landsat data was reclassified into a 5-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.



3.5 Exercise

Exercise 1 (Easy): Lucas County in Northwest Ohio wants to expand its urban forest network to double its current size by 2050. To reach this goal, the city government plans on converting some of the existing open space throughout the city into forest. For this exercise, reclassify water and wetlands classes into 1, forest data (evergreen, deciduous, mixed) into 2, open space developed land into 3, and all other land cover uses into 0. Use the NLCD Land Cover Legend attached below.

https://www.mrlc.gov/data/legends/national-land-cover-database-class-legend-and-description