5.0 [Actual Step #3] Reprojecting Raster Data

By David Gerstenfeld, Adrian Terech

Introduction

In this chapter, we will go over the process of reprojecting raster datasets to a different CRS. This step is essential in optimization and data cleaning to ensure that researchers are getting the most accurate results in their studies. The first part of this chapter will cover how to project a raster dataset to a defined CRS. The second part will cover the use of looping to reproject multiple raster datasets and the use of source rasters as guidelines on reprojection.

5.1 Defining CRS and Variables

Before reprojecting, it is essential to determine what CRS you are planning to reprojection a raster dataset to. By using sites like EPSG.io, you can determine the EPSG code for your raster dataset and define it using:

```
dst_crs = 2272 (This is the EPSG code for NAD 1983 State Plane Pennsylvania South, the CRS used for this study
```

The section below opens the raster dataset, in this case the land cover dataset, and defines a wide variety of variables that will be used to reproject the raster dataset. Once the land cover dataset is opened, it is redefined as variable *raster*. The 5 variables defined from the contents of the land cover raster include *src_shape*, *raster_data*, *src_dtype*, *src_transform*, and *src_crs*. Variable *src_shape* is defined by collecting the height and width of the raster dataset. *Raster_data* is defined by reading and collecting the first band, which is then used to determine the variable *src_dtype* by observing the data format used to visualize the first band. *Src_transform* is defined by the affine transformation of the dataset, which <>. And lastly, variable *src_crs* is defined by the land cover raster's existing CRS.

```
with rasterio.open(land_cover) as raster:
    src_shape = (raster.height, raster.width)
    raster_data = raster.read(1)
    src_dtype = raster_data.dtype
    src_transform = raster.transform
    src_crs = raster.crs
```

5.2 Reprojecting the raster

Once the necessary variables are defined, new reprojection variables need to be defined under the *dst_crs*. This process is conducted using the **rasterio.warp.calculate_default_transform** function. This function converts a raster dataset to a newly defined CRS, which we have done at the beginning with variable *dst_crs*. The *dst_crs* is added into the calculate_default_transform function alongside variables like *src_crs*, *raster.width*, *raster.height*, and *raster.bounds* to define three new variables.

```
dest_transform, dest_width, dest_height = rasterio.warp.calculate_default_transform(
    src_crs, dst_crs, raster.width, raster.height, *raster.bounds)
```

The three new variables *dest_transform*, *dest_width*, and *dest_height* are all defined by reprojecting the width, height, and bounding box of the raster dataset into the new CRS. After the three variables are defined, an empty destination array named *destination* is created to contain the height and width of the reprojected raster using the defined data type from the original raster dataset. Listed below is the destination array:

```
destination = np.empty((dest_height, dest_width), dtype=src_dtype)
```

After the destination array is created, the reprojection process can be completed. Using the **reprojection** function, the first band is entered into the function under variable *source*. The recently created destination array is also added into the reprojection under the same name. From both the original raster and reprojected rasters, the Affine transformation and CRS are added into the reprojection function under variables *src_transform*, *src_crs*, *dest_transform*, and *dst_crs*. Lastly, the variable *resampling* is defined by using the *Resampling* function to resample the raster pixel values based on their nearest neighbors. Other resampling methods including bilinear and cubic (using either **Resampling.bilinear** or **Resampling.cubic**) can be substituted to produce more accurate results, but will take longer to process.

```
reproject(
    source=raster_data,
    destination=destination,
    src_transform=src_transform,
    src_crs=src_crs,
    dest_transform=dest_transform,
    dst_crs=dst_crs,
    resampling=Resampling.nearest
)
```

The final step in reprojection is saving the results from the **reproject** function into an output file. Much like in previous steps in the reprojection process, multiple variables are utilized when the output file *landcover_reprojected* is opened. Many of the reprojected variables determined in previous steps are saved into the output file including height, width, CRS, and Transformation. The one exception is the data type as we want to keep the same data type from the original raster for precision. The variable *count* is determined based on how many bands the raster dataset contains. Most raster datasets contain only 1 band, hence in most cases the count will remain 1. But in rare instances where RBG or multi-spectral imaging is used, the variable will need to be changed to either 2 or 3. Lastly, the data will be saved by adding the destination array into the output raster by using *dest.write*. The number in this function once again will be determined based on how many bands the raster contains.

```
with rasterio.open(
    landcover_reprojected,
    'w',
    driver='GTiff',
    height=dest_height,
    width=dest_width,
    count=1,
    dtype=src_dtype,
    crs=dst_crs,
    transform=dest_transform
) as dest:
    dest.write(destination, 1)
```

5.3 Reprojection Looping

There are some additional ways to optimize the reprojection process, especially when dealing with multiple rasters. One method is to use looping to reproject multiple rasters at once. To complete a reprojection loop, the original code does need to be altered in a few places. The first change is the creation of two additional variables at the start of the script: *input_raster*, and *output_raster*.

```
input raster = input_file_path1, input_file_path2, ….
output raster = output_file_path1, output_file_path2, ….
```

For organization purposes, it would be recommended to add these two variables after the reprojected CRS has been defined and before the raster dataset is read.

A loop now needs to be constructed just before the raster dataset is opened. This can be accomplished by using a for loop that combines the input_raster and output_raster paths into a pair. For this reason, it is essential to ensure that each of the file paths match with each other (for example: land_cover (input) and land_cover_reprojected (output)).

```
for input_raster, output_raster in zip(input_rasters, output_rasters):
```

The with loop used earlier in the chapter is used below the for loop with minimal changes, with variable *input_raster* being opened and defined in this case.

```
with rasterio.open(input_raster) as raster:
    src_shape = (raster.height, raster.width)
    raster_data = raster.read(1)
    src_dtype = raster_data.dtype
    src_transform = raster.transform
    src_crs = raster.crs
```

Throughout the rest of the reprojection code, the steps remain the same until the end where the reprojected data is saved into an output file. Use variable *output_raster* when opening the output file to successfully complete the loop.

```
with rasterio.open(
    output_raster,
    'w',
    driver='GTiff',
    height=dest_height,
    width=dest_width,
    count=1,
    dtype=src_dtype,
    crs=dst_crs,
    transform=dest_transform
) as dest:
    dest.write(destination, 1)
```

5.4 Source Rasters

An additional method that can be used is using source rasters. Source rasters are raster datasets that have already been reprojected and can be used to reproject another raster dataset. The use of source rasters can help speed up the process of reprojection by granting the user the ability to bypass using the **calculate_default_transform** function. To replace this function, both the source raster and the raster that is going to be reprojected will need to be read. Let's suppose that in this case, *land_cover* has already been reprojected. We will use it as a source raster to reproject *tree_cover* to the same projection as *land_cover*. In this section, *land_cover* will be redefined as *src*, while *tree_cover* will be redefined as *target*.

```
with rasterio.open(land_cover) as source:
    source_shape = (source.height, source.width)
    source_data = source.read(1)
    source_dtype = source_data.dtype
    source_transform = source.transform
    source_crs = source.crs

with rasterio.open(tree_cover) as target:
    target_shape = (target.height, target.width)
    target_data = target.read(1)
    target_dtype = target_data.dtype
    target_transform = target.transform
    target_crs = target.crs
```

Since the **calculate_default_transform** function is no longer being used, we can skip that section. The destination array when using a source raster should use the height and width from the source raster, while the target raster's data type should be used. For example:

```
destination = np.empty((source_height, source_width), dtype=target_dtype)
```

The **reprojection** function should also reflect these changes, where the source raster's transformation and CRS should be used as the reprojected data. The target raster's data type, transformation, and CRS should be used as the original raster datasets as shown below:

```
reproject(
    source=target_data,
    destination=destination,
    target_transform=target_transform,
    target_crs=target_crs,
    reproject_transform=source_transform,
    reproject_crs=source_crs,
    resampling=Resampling.nearest
    )
```

Saving the newly reprojected raster file is similar in this instance, with minor changes made to reflect the variable names used in this section of code. Note that the height and width used for the reprojected raster is from the source raster's height and width defined earlier in the code.

```
 with rasterio.open(
    treecover_reprojected,
      'w',
    driver='GTiff',
    height=source_height,
    width=source_width,
    count=1,
    dtype=target_dtype,
    crs=reproject_crs,
    transform=reproject_transform
) as dest:
    dest.write(destination, 1)
```

It is possible to use both reprojection looping and source rasters in the same script. This can be done by combining the steps mentioned above in section 5.3 and 5.4 by defining input and output paths, reading the source raster, and using reprojection looping to conduct the reprojection process. Also note that in this case, the **calculate_default_transform** should be removed from the reprojection loop and the variables be defined to reflect the steps used in section 5.4.

5.5 Exercises

Exercise 1 (Easy): The City of Philadelphia is planning on conducting a study on how much of the city is covered by car infrastructure. A raster dataset containing impervious surface cover will be used, however the dataset is in a different projection than the city's other datasets. Convert the Impervious Surface Cover dataset into NAD 1983 State Plane Pennsylvania South.

Exercise 2 (Advanced): On the other side of the Delaware, Camden is getting ready to conduct a study on Urban Heat Island using Land Cover and Tree Cover datasets. However, both of the rasters are not set to the default reference system used by the city. Using one of the raster datasets already in their directory, write a loop that converts the two raster datasets into WGS 84 UTM / Zone 18N. Use EPSG.io to search for the EPSG code of WGS 84 / UTM Zone 18N.