# **Exercise: Multi-Ring Buffers and Spatial Joins**

## Summary

This exercise builds a multi-ring set of buffers around a highway and then uses a spatial join to determine which property tax parcels are within each ring.

## Input Data

There are two input files: the geopackage file **onondaga.gpkg** created in the previous assignment and **onondaga-tax-parcels.gpkg**, a geopackage file with a range of information, including the centroid, assessed value, and other characteristics, about each tax parcel in Onondaga County. It should be downloaded from the course Google Drive.

#### **Deliverables**

There are three deliverables: a script called **parcels.py**, a QGIS project file called **rings.qgz**, and an image called **rings.png**.

#### Instructions

#### A. Script parcels.py

- 1. Import geopandas as gpd.
- 2. Use gpd.read\_file() to read the dissolved interstates layer from "onondaga.gpkg" into a new variable called interstates. To be sure you have the right layer, include the argument layer="interstates" in the call.
- 3. Create a list called radius that contains the following numbers: 200, 400, 600, 800, 1000, 1200, 1400, 1600 and 3200. These will be the outer radii, in meters, of the rings we'll create. There will be several near the highway and then a broader ring to select parcels that are further away and could be used as a comparison group. Also, that ring demonstrates that the radii of the rings can vary.
- 4. Now we'll build the geometries of the rings. Start by creating a new empty list called <code>geo\_list</code> to contain them.
- 5. Next, create a variable called <code>last\_buf</code> and set it equal to <code>None</code>. As you'll see below, we'll create the buffers moving outward from the interstate. This variable will be used to make the buffers into rings by allowing us to subtract out the previous buffer when building the next one.
- 6. Use loop variable r to loop over radius. Within the loop, do the following:
  - 1. Set this\_buf to the result of calling the .buffer() method on interstates with argument r.
  - 2. Use an if statement to see if the length of <code>geo\_list</code> is 0, which indicates that no rings have been built yet. If that's true, inside the if block add a line to do the following:
    - 1. Append this\_buf[0] to geo\_list. Don't overlook the [0]: that's needed to extract the ring's geometry from this\_buf, which is a the GeoSeries.
  - 3. Handle other cases using an else statement with a two-line block of code that does the following:
    - 1. Creates a variable called change that is equal to the result of calling the .difference() method on this\_buf with the argument last\_buf. The result will be the ring buffer for the current value of r: that is, the area within r meters from the interstate but outside the previous buffer.
    - 2. Append change[0] to geo\_list . Remember the [0] again.

- 4. After the end of the else block, but still inside the loop, set last\_buf equal to this\_buf. To be clear, this line should *not* be in the if / else block: it needs to be outside that so it is executed each trip through the loop no matter what happens with the if statement.
- 7. At this point the for loop is complete. Now, after then end of the loop, set ring\_layer equal to the result of calling gpd.GeoDataFrame() with argument geometry=geo\_list to create a new GeoDataFrame with the ring geometry just built.
- 8. Set the CRS of ring\_layer by setting ring\_layer to the result of calling .set\_crs() on ring\_layer using argument interstates.crs. This copies the CRS of interstates to ring\_layer so the coordinates in the ring\_layer polygons can be interpreted correctly. Note that in this case the call is .set\_crs() NOT .to\_crs() . (FAQ 1)
- 9. Create a new column called "radius" in ring\_layer that is equal to the radius variable. The column will be part of the attribute table of the layer we're building and will indicate the outer radius of each ring.
- 10. Save ring\_layer to a geopackage file called "near-parcels.gpkg" as layer "rings".
- 11. Read "onondaga-tax-parcels.gpkg" into a geodataframe called parcels. The file has a bit more than 180,000 records and quite a few fields so don't be surprised if it takes some time to load.
- 12. Now we'll do the spatial join to add the ring information onto the parcel layer. Create variable near by calling .sjoin() on parcels with three arguments: ring\_layer, how="left", and predicate="within".

  The how="left" has a special meaning with spatial joins: it indicates that the new dataframe should use the geometry from the left dataset (the parcels).
- 13. Check the join by printing the result of applying .value\_counts() to the "radius" column of near using dropna=False. As a check, you should have 6247 parcels in the 200 m ring and a bit under 68,000 beyond the 3200 m ring (NaN for the radius).
- 14. Save near to geopackage "near-parcels.gpkg" as layer "parcels"
- 15. For convenience in a subsequent exercise, also write out the attribute table as a CSV file. To do so, drop the "geometry" column from near and then use .to\_csv() to write the revised version of near out as "near-parcels.csv" using index=False.

#### B. Files rings.qgz and rings.png

- 1. Start a new QGIS project and load in the county and interstates layers from "onondaga.gpkg".

  Don't load the buffer layer since we'll be using the rings instead.
- 2. Next, load in the rings and parcels layers from "near-parcels.gpkg".
- 3. Stack the layers so the parcels are on the top, then the interstates, the rings, and then the county. Use "Categorized" as the style of the parcels, with the value set to "radius" and the color ramp set to "Magma" or whatever alternative you prefer.
- 4. Save the project as "rings.qgz" and export the map as "rings.png".

## Tips

Parcels outside the largest ring will have missing data for their ring number.

### **FAQs**

1. The .set\_crs() call is used to indicate the coordinate system of data stored in a GeoSeries or GeoDataFrame and it doesn't change the coordinates at all. It's used for building new GeoSeries or GeoDataFrames

from scratch (as is done here). The .to\_crs() command is **very** different: it converts all the geographic coordinates in the object to from their existing coordinate system to a new one. As a result, it should only be called on objects that have a CRS.