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.

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.
2. Read the 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 variable called `dissolved` by dissolving `interstates` the same way it was done in the previous assignment.
4. Change the CRS for the dissolved layer to EPSG:26918 by setting `dissolved` to the value returned by calling the `.to_crs()` method of `dissolved` with `epsg=26918`. That matches the projection used by Onondaga County, which will speed things up later. It is a slightly different specification for UTM 18N from the one used in the last assignment.
5. Create a list called `radius` that contains the following numbers: 200, 400, 600, 800, 1000 and 2000. 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. In part, the last ring is there to emphasize that the radii of the rings can vary.
6. Set `ring_layer` equal to the result of calling `geopandas.GeoDataFrame()` to create a new, empty geodataframe.
7. 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.
8. Now we'll build the geometries of the rings. Start by creating a new empty list called `geo_list` to contain them.
9. Next, create a variable called `last_buf` and set it equal to `None`. 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.
10. Use running 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 `dissolved` with argument `r`.
    2. Use an `if` statement to see if the length of `geo_list` is 0, which indicates that no rings have been built yet. If that's true, do the following:
        1. Append `this_buf[0]` to `geo_list`.

    3. Handle other cases using an `else` statement with a 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`.
    4. After the end of the `else` block set `last_buf` equal to `this_buf`.
11. After the end of the `for` loop, add a column called `geometry` to `ring_layer` and set it equal to `geo_list`. The effect will be to load the geometry (polygons) for each ring into the layer.
12. Set the CRS of the ring layer by setting `ring_layer` to the result of calling `.set_crs()` on `ring_layer` using argument `epsg=26918`.
13. Save `ring_layer` to a geopackage file called `"rings.gpkg"` as layer `"rings"`.
14. Read `"onondaga-tax-parcels.gpkg"` into a geodataframe called `parcels`. The file has a lot of records and quite a few fields so don't be surprised if this and the next few steps are a little slow.
15. Now we'll do the spatial join to add the ring information onto the parcel layer. Create variable `near` by calling `geopandas.overlay()` with three arguments: `parcels`, `ring_layer`, and `how="intersection"`.
16. Save `near` to geopackage `"near-parcels.gpkg"` as layer `"parcels"`.
17. 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"`.
2. Next, load in `"rings.gpkg"`.
3. Now add the tax parcel centroids for the rings by loading `"near-parcels.gpkg"`.
4. 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 "random" or whatever alternative you prefer.
5. Save the project as `"rings.qgz"` and export the map as `"rings.png"`.

Tips

- You'll notice right away that the only parcels in `"near-parcels.gpkg"` are those that fall within one of the rings: parcels outside the outer ring are not included. That's because the `how="intersection"` directive to the overlay method does an inner join and only returns records for items with geometries (parcel centroid and ring) that intersect. Including the other parcels in the output could be done by changing `how="intersection"` to `how="union"`. All of the original parcels would appear in the output but those outside the largest ring would end up with missing values for `"radius"`. It would take considerably longer, however.