Data exploration with GeoPandas

CS424: Visualization & Visual Analytics

Fabio Miranda

https://fmiranda.me

UIC COMPUTER SCIENCE

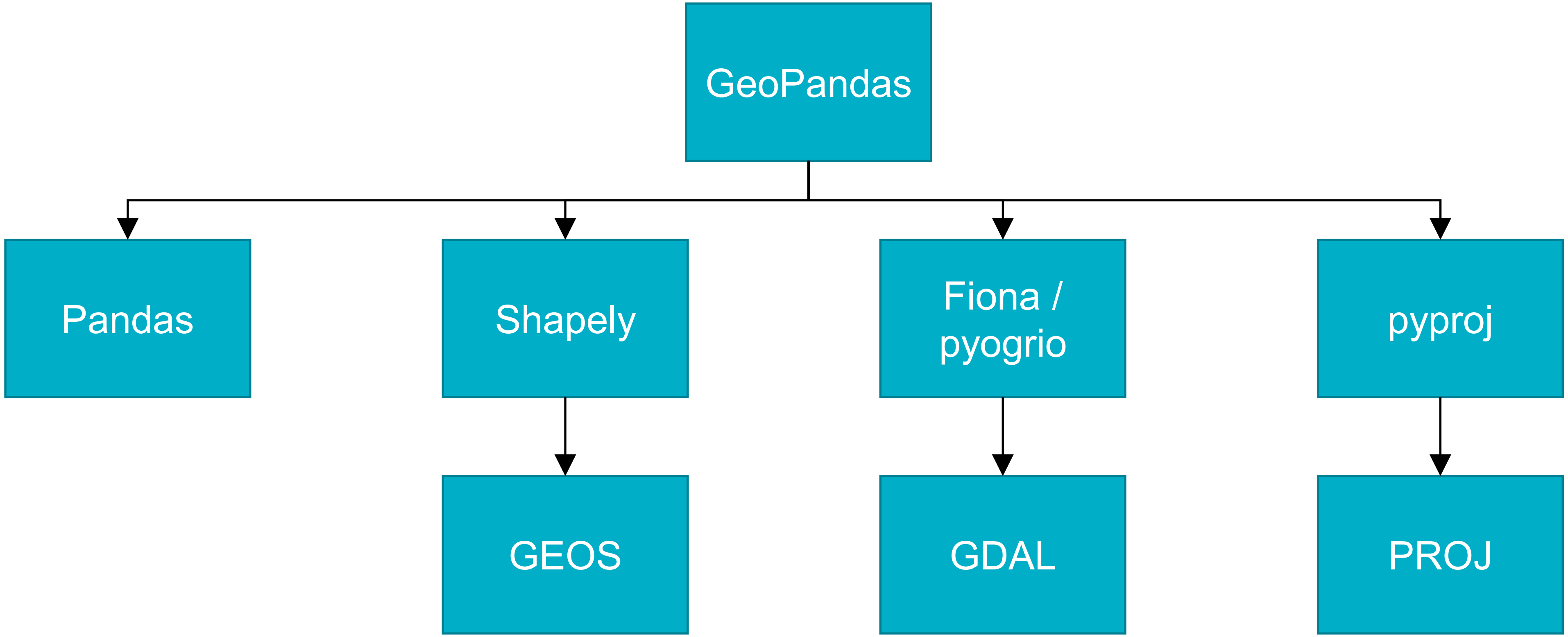
GeoPandas

- Geospatial data & Python made easier.
- Extends Pandas to work with geographic objects and spatial operations.
- Combines the power of several libraries (Pandas, geos, shapely, gdal, pypro j, rtree, …)

GeoPandas

- Read and write several geo formats (Fiona, GDAL).
- Usual DataFrame manipulation.
- Element-wise spatial operations (intersection, union, difference, …)
- Re-project data.
- Visualize geometries.
- Advanced spatial operations: spatial joins and overlays.

GeoPandas


GeoPandas
Pandas
Shapely
Fiona / pyogrio
pyproj
GEOS
GDAL
PROJ

Shapely

- Python package for the manipulation of geometric objects.

```
from shapely.geometry import Point, LineString, Polygon

point = Point(1, 1)
line = LineString([(0, 0), (1, 2), (2, 2)])
poly = line.buffer(1)
```

```
line
```

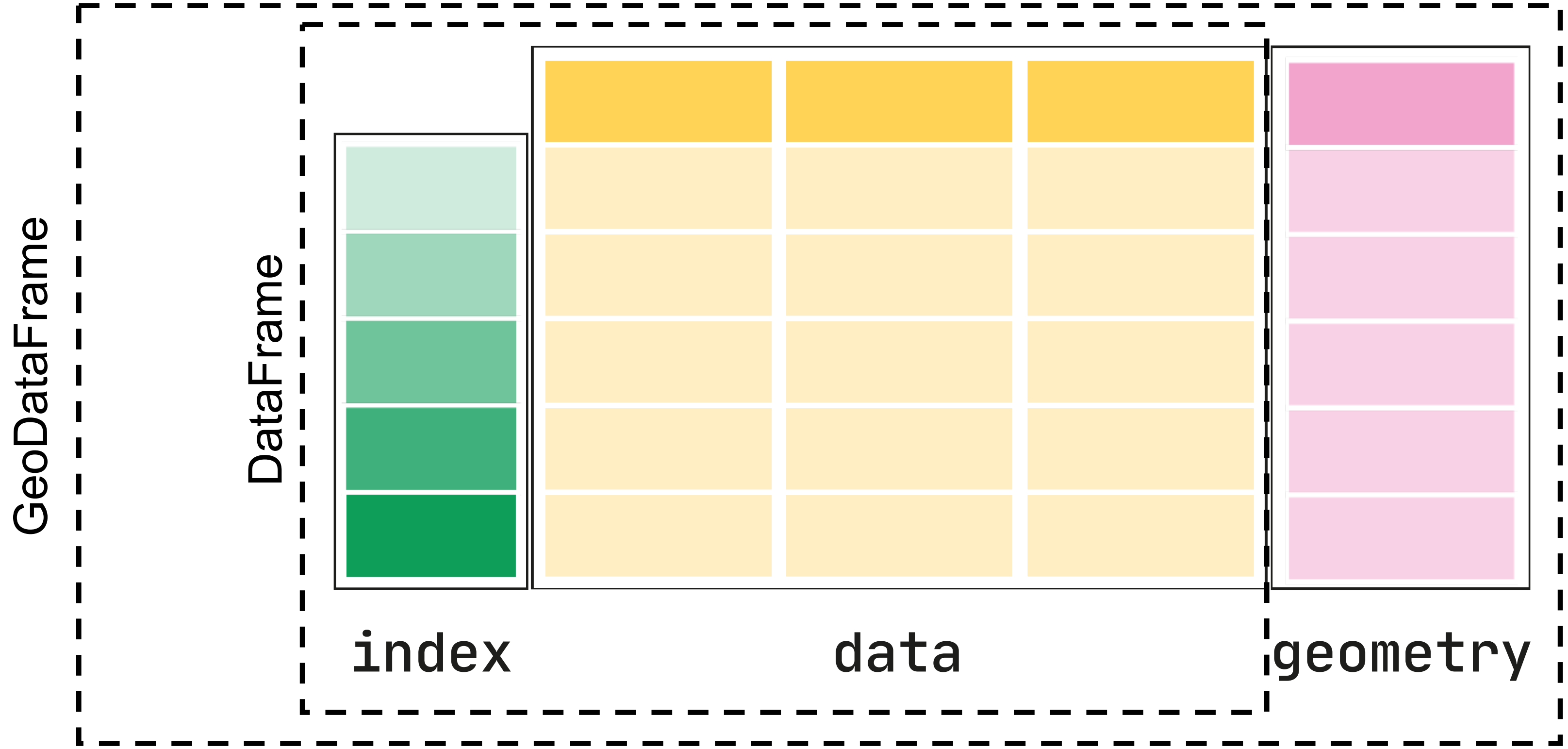
```
poly
```

```
poly.contains(point)
```

```
True
```

GeoPandas

- Core data structure in GeoPandas is the GeoDataFrame (subclass of Pandas' DataFrame).
  - It can store geometry columns and perform spatial operations.

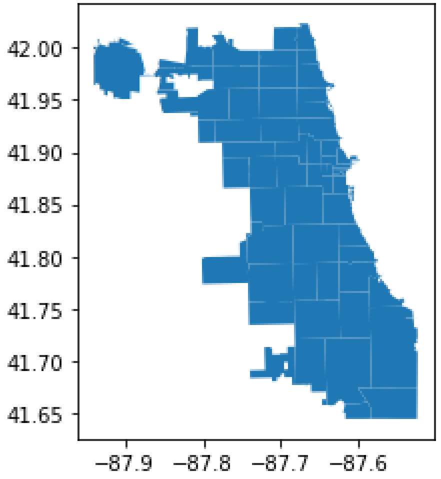


Reading and writing files

```
gdf = gpd.read_file('chicago.geojson')
```

```
gdf.plot()
```

<AxesSubplot:>




```
gdf
```

	objectid	shape_area	shape_len	zip	geometry
0	33	106052287.488	42720.0444058	60647	MULTIPOLYGON (((-87.67762 41.91776, -87.67761 ...
1	34	127476050.762	48103.7827213	60639	MULTIPOLYGON (((-87.72683 41.92265, -87.72693 ...
2	35	45069038.4783	27288.6096123	60707	MULTIPOLYGON (((-87.78500 41.90915, -87.78531 ...
3	36	70853834.3797	42527.9896789	60622	MULTIPOLYGON (((-87.66707 41.88885, -87.66707 ...
4	37	99039621.2518	47970.1401531	60651	MULTIPOLYGON (((-87.70656 41.89555, -87.70672 ...
...	...	...	...	...	...
56	57	155285532.005	53406.9156168	60623	MULTIPOLYGON (((-87.69479 41.83008, -87.69486 ...
57	58	211114779.439	58701.3253749	60629	MULTIPOLYGON (((-87.68306 41.75786, -87.68306 ...
58	59	211696050.967	58466.1602979	60620	MULTIPOLYGON (((-87.62373 41.72167, -87.62388 ...
59	60	125424284.172	52377.8545408	60637	MULTIPOLYGON (((-87.57691 41.79511, -87.57700 ...
60	61	167872012.644	53040.9070778	60619	MULTIPOLYGON (((-87.58592 41.75150, -87.58592 ...

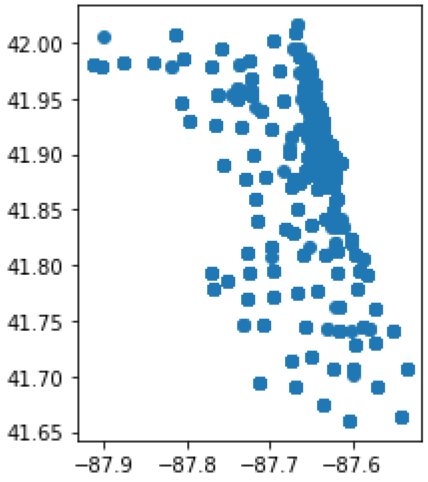
61 rows × 5 columns

Reading CSV file

```
df = pd.read_csv('data/Taxi_Trips.csv')
geometry = [Point(xy) for xy in zip(df['Pickup Centroid Longitude'], df['Pickup Centroid Latitude'])]
gdf = gpd.GeoDataFrame(df, geometry=geometry, crs=4326)
```

```
gdf.plot()
```

```
<AxesSubplot:>
```



Geometry from lon / lat (x / y)

Projections

```
gdf
```

	objectid	shape_area	shape_len	zip	geometry
0	33	106052287.488	42720.0444058	60647	MULTIPOLYGON (((-87.67762 41.91776, -87.67761 ...
1	34	127476050.762	48103.7827213	60639	MULTIPOLYGON (((-87.72683 41.92265, -87.72693 ...
2	35	45069038.4783	27288.6096123	60707	MULTIPOLYGON (((-87.78500 41.90915, -87.78531 ...
3	36	70853834.3797	42527.9896789	60622	MULTIPOLYGON (((-87.66707 41.88885, -87.66707 ...
4	37	99039621.2518	47970.1401531	60651	MULTIPOLYGON (((-87.70656 41.89555, -87.70672 ...
...	...	...	...	...	...
56	57	155285532.005	53406.9156168	60623	MULTIPOLYGON (((-87.69479 41.83008, -87.69486 ...
57	58	211114779.439	58701.3253749	60629	MULTIPOLYGON (((-87.68306 41.75786, -87.68306 ...
58	59	211696050.967	58466.1602979	60620	MULTIPOLYGON (((-87.62373 41.72167, -87.62388 ...
59	60	125424284.172	52377.8545408	60637	MULTIPOLYGON (((-87.57691 41.79511, -87.57700 ...
60	61	167872012.644	53040.9070778	60619	MULTIPOLYGON (((-87.58592 41.75150, -87.58592 ...

61 rows × 5 columns

Projecting to EPSG:3395

```
gdf = gdf.to_crs("EPSG:3395")
```

```
gdf
```

	objectid	shape_area	shape_len	zip	geometry
0	33	106052287.488	42720.0444058	60647	MULTIPOLYGON (((-9760228.181 5120114.708, -976...
1	34	127476050.762	48103.7827213	60639	MULTIPOLYGON (((-9765706.326 5120843.341, -976...
2	35	45069038.4783	27288.6096123	60707	MULTIPOLYGON (((-9772181.764 5118831.519, -977...
3	36	70853834.3797	42527.9896789	60622	MULTIPOLYGON (((-9759053.446 5115807.386, -975...
4	37	99039621.2518	47970.1401531	60651	MULTIPOLYGON (((-9763449.188 5116805.817, -976...
...	...	...	...	...	...
56	57	155285532.005	53406.9156168	60623	MULTIPOLYGON (((-9762139.685 5107055.000, -976...
57	58	211114779.439	58701.3253749	60629	MULTIPOLYGON (((-9760833.554 5096312.536, -976...
58	59	211696050.967	58466.1602979	60620	MULTIPOLYGON (((-9754228.915 5090933.835, -975...
59	60	125424284.172	52377.8545408	60637	MULTIPOLYGON (((-9749017.532 5101851.550, -974...
60	61	167872012.644	53040.9070778	60619	MULTIPOLYGON (((-9750019.820 5095367.709, -975...

61 rows × 5 columns

Projections

```
1 gdf
```

	objectid	shape_area	shape_len	zip	geometry
0	33	106			.67761 ...
1	34	127			.72693 ...
2	35	450			.78531 ...
3	36	708			.66707 ...
4	37	990			.70672 ...
...	...				...
56	57	155			.69486 ...
57	58	21			.68306 ...
58	59	211			.62388 ...
59	60	125424284.172	52377.8545408	60637	MULTIPOLYGON (((-87.57691 41.79511, -87.57700 ...
60	61	167872012.644	53040.9070778	60619	MULTIPOLYGON (((-87.58592 41.75150, -87.58592 ...

```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

61 rows × 5 columns

Projecting to EPSG:3395

```
1 gdf = gdf.to_crs("EPSG:3395")
```

```
1 gdf
```

	objectid	shape			netry
0	33	1060522			976...
1	34	1274760			976...
2	35	4506903			977...
3	36	7085383			975...
4	37	9903962			976...
...	...				...
56	57	1552855			976...
57	58	2111147			976...
58	59	2116960			975...
59	60	1254242			974...
60	61	167872012.644	53040.9070778	60619	MULTIPOLYGON (((-9750019.820 5095367.709, -975...

```
<Derived Projected CRS: EPSG:3395>
Name: WGS 84 / World Mercator
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: World between 80°S and 84°N.
- bounds: (-180.0, -80.0, 180.0, 84.0)
Coordinate Operation:
- name: World Mercator
- method: Mercator (variant A)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

61 rows × 5 columns

Simple operations

- Accessing & plotting geometry area

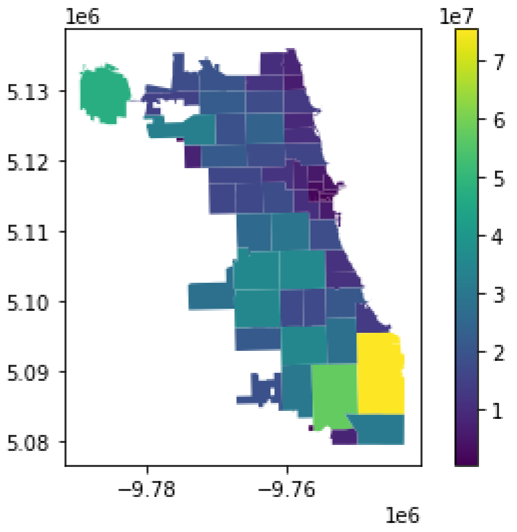
```
gdf.area
```

```
0     1.774279e+07
1     2.132685e+07
2     7.540024e+06
3     1.184732e+07
4     1.656003e+07
          ...
56    2.592093e+07
57    3.515998e+07
58    3.521726e+07
59    2.089192e+07
60    2.793029e+07
Length: 61, dtype: float64
```

```
gdf['area'] = gdf.area
```

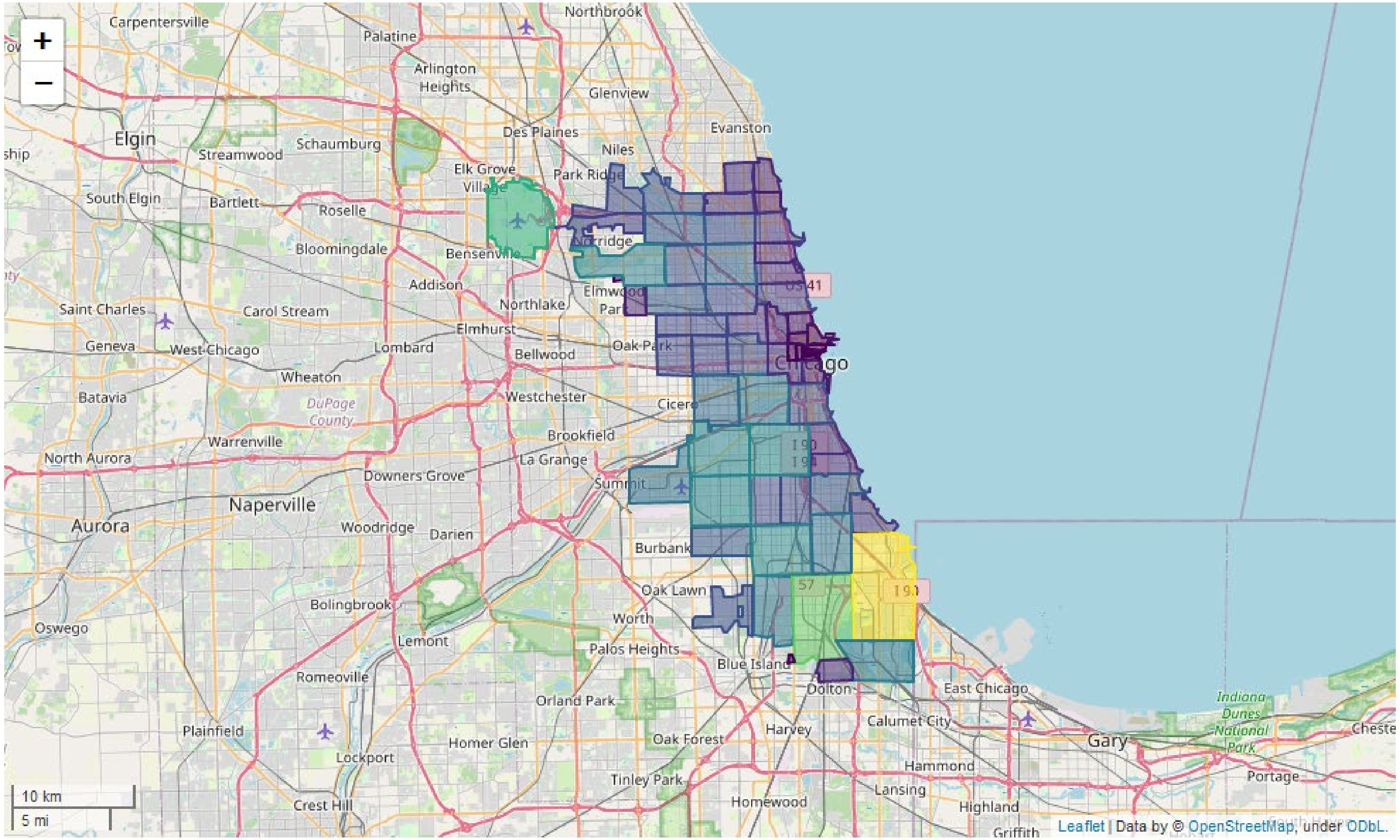
```
gdf.plot("area", legend=True)
```

```
<AxesSubplot:>
```



Simple operations

```
gdf.explore("area", legend=False)
```



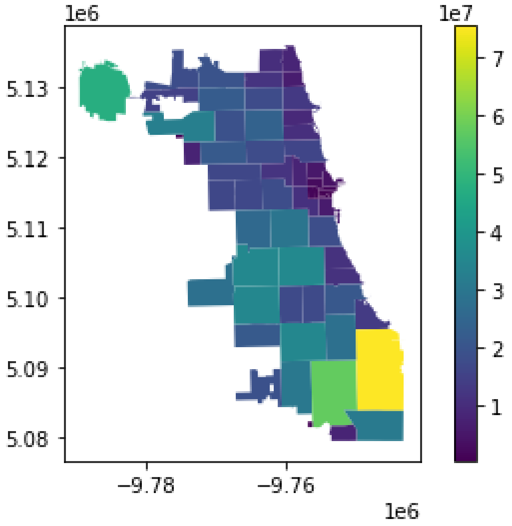
Simple operations

Changing geometry

```
gdf['area'] = gdf.area
```

```
gdf.plot("area", legend=True)
```

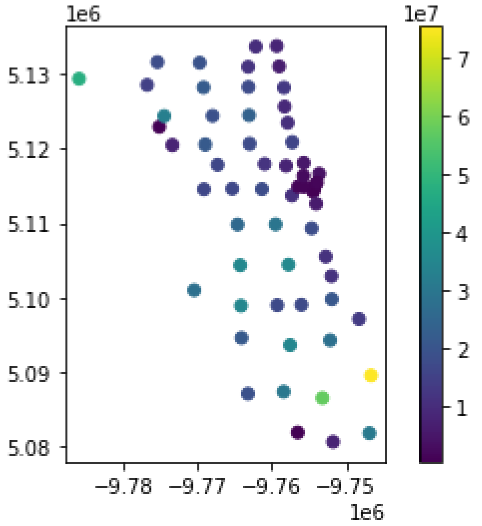
<AxesSubplot:>




```
gdf['centroid'] = gdf.centroid
```

```
gdf = gdf.set_geometry('centroid')
gdf.plot("area", legend=True)
```

<AxesSubplot:>



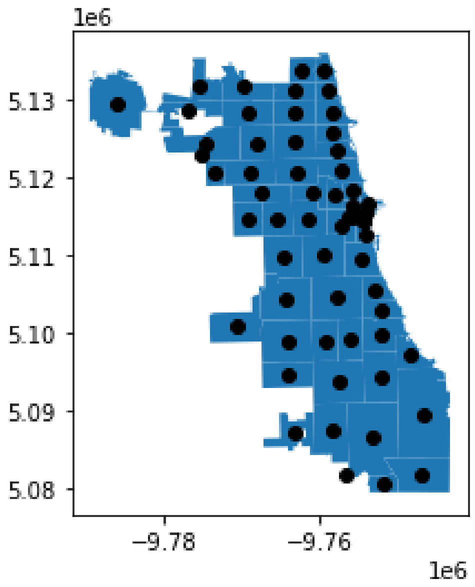
Simple operations

- Plotting centroids and zip areas:

```
gdf = gdf.set_geometry('geometry')
ax = gdf['geometry'].plot()
gdf['centroid'].plot(ax=ax, color="black")
```

Setting geometry back

```
<AxesSubplot:>
```



Geometry operations

```
gdf["convex_hull"] = gdf.convex_hull
```

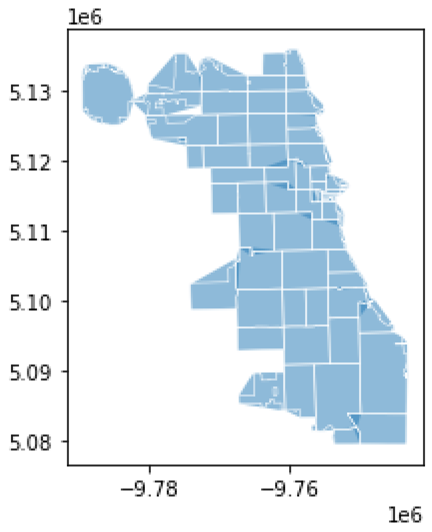
New column with convex hull

```
gdf['boundary'] = gdf.boundary
```

New column with boundary (i.e., outline)

```
ax = gdf["convex_hull"].plot(alpha=.5)
gdf['boundary'].plot(ax=ax, color="white", linewidth=.5)
```

<AxesSubplot:>



Geometry operations

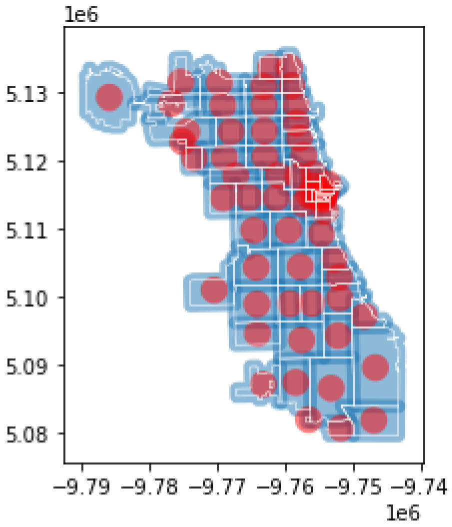
```
gdf["buffered"] = gdf.buffer(1000)
gdf["buffered_centroid"] = gdf["centroid"].buffer(2000)
```

Creating buffer around zip areas

Creating buffer around centroids

```
ax = gdf["buffered"].plot(alpha=.5)
gdf["buffered_centroid"].plot(ax=ax, color="red", alpha=.5)
gdf["boundary"].plot(ax=ax, color="white", linewidth=.5)
```

```
<AxesSubplot:>
```



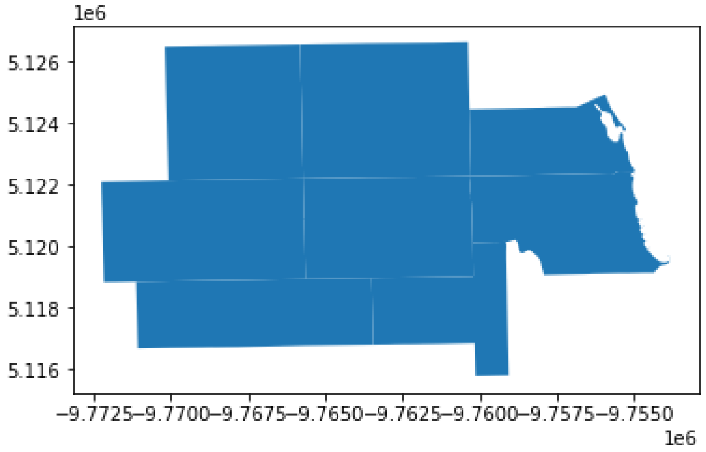
Geometry relations

- GeoPandas offers a series of operations for geometry relations:
  - Crosses, intersects, overlaps, covers, within, touches, …

```python
selected = gdf.iloc[0]['geometry']
```

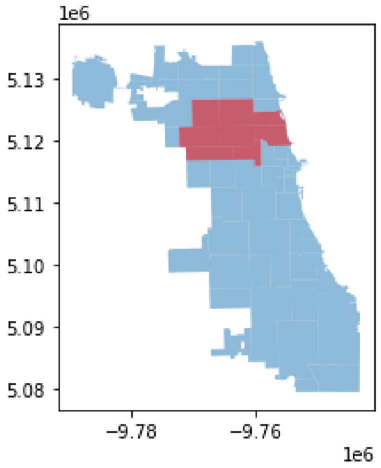
```python
gdf[gdf['buffered'].intersects(selected)].plot()
```

<AxesSubplot:>




```python
intersected = gdf[gdf['buffered'].intersects(selected)]
ax = gdf.plot(alpha=.5)
intersected.plot(ax=ax, color="red", alpha=.5)
```

<AxesSubplot:>



Geometry relations

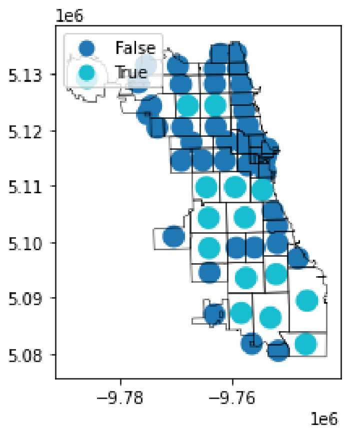
```
gdf["within"] = gdf["buffered_centroid"].within(gdf)
gdf["within"]
```

Whether buffered centroid is within zip area

```
0     False
1     False
2     False
3     False
4     False
      ...
56     True
57     True
58     True
59    False
60     True
Name: within, Length: 61, dtype: bool
```

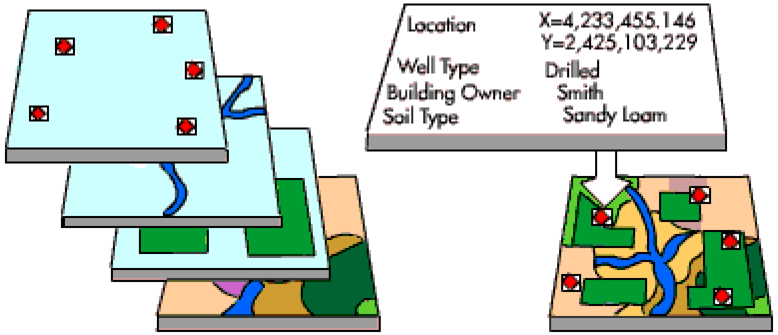
```
gdf = gdf.set_geometry("buffered_centroid")
ax = gdf.plot("within", legend=True, categorical=True, legend_kwds={'loc': "upper left"})
gdf["boundary"].plot(ax=ax, color="black", linewidth=.5)
```

```
<AxesSubplot:>
```



Spatial joins

- A spatial join combines two GeoDataFrames based on the spatial relationship between their geometries.
- Example: spatial join between point layer (e.g., taxi pickups) and a polygon layer (e.g., zip codes).



Spatial join

```
x_min, y_min, x_max, y_max = gdf.total_bounds

n = 1000

x = np.random.uniform(x_min, x_max, n)
y = np.random.uniform(y_min, y_max, n)

gdf_points = gpd.GeoDataFrame(geometry=gpd.points_from_xy(x, y), crs=gdf.crs)
gdf_points = gdf_points[gdf_points.within(gdf.unary_union)]
```

Bounds of GeoDataFrame

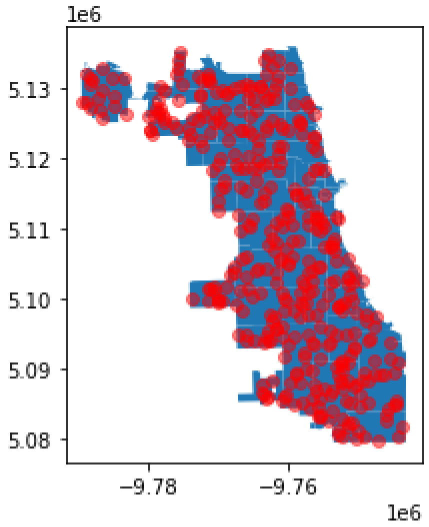
Sample size

Creating GeoDataFrame

Only keeping points within polygons

```
ax = gdf.plot()
gdf_points.plot(ax=ax, color="red", alpha=.5)
```

<AxesSubplot:>



Spatial join

- Spatial join: for each point, is it *within* what zip code?

```
gpd.sjoin(gdf_points, gdf, predicate='within')
```

	geometry	index_right	objectid	shape_area	shape_len	zip
1	POINT (-9774274.365 5122584.288)	51	52	194062612.162	77647.3180069	60634
133	POINT (-9774348.810 5124145.541)	51	52	194062612.162	77647.3180069	60634
164	POINT (-9776525.134 5125184.977)	51	52	194062612.162	77647.3180069	60634
207	POINT (-9770521.404 5126432.046)	51	52	194062612.162	77647.3180069	60634
253	POINT (-9777824.534 5126280.441)	51	52	194062612.162	77647.3180069	60634
...	...	...	...	...	...	...
578	POINT (-9761501.347 5118140.579)	3	36	70853834.3797	42527.9896789	60622
917	POINT (-9763366.621 5118966.647)	3	36	70853834.3797	42527.9896789	60622
714	POINT (-9755003.744 5114813.246)	40	26	4847124.8171	14448.1749926	60602
756	POINT (-9772253.752 5121497.783)	2	35	45069038.4783	27288.6096123	60707
894	POINT (-9759652.063 5132688.902)	8	1	49170578.9623	33983.9133065	60626

392 rows × 6 columns

Spatial join

- Grouping by zip code value to obtain number of points within that area.

```
result = gpd.sjoin(gdf_points, gdf, predicate='within').groupby('zip').count()
```

Grouping by zip code

```
result = result.filter(['geometry'])
result = result.rename(columns={'geometry': 'count'})
```

```
result
```

zip	count
60602	1
60605	1
60607	7
60608	12
60609	11
60610	3
60612	4
60613	4
60614	2
60615	8
60616	12
60617	20

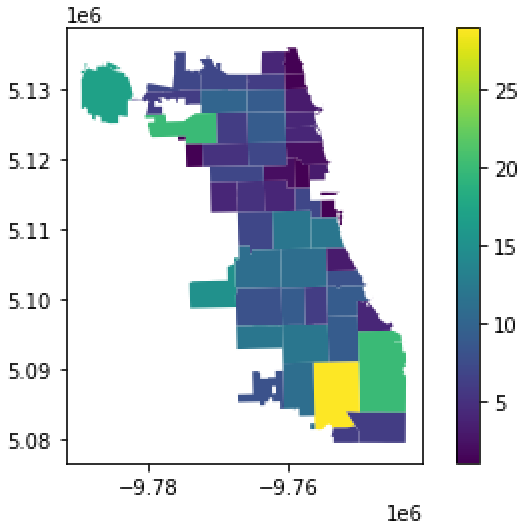
Spatial join

```
merged = pd.merge(result, gdf, right_on='zip', left_index=True)
```

Merging with previous zip code GeoDataFrame

```
merged = merged.set_geometry('geometry')
merged.plot('count', legend=True)
```

<AxesSubplot:>



Spatial join

Aggregating points over spatial regions:

1. Spatial join: map between point and polygon.
2. Group by: aggregate (sum, count, mean, …) by polygon.
3. Merge / join: map between aggregations and polygons.