12.1_geopandas

November 6, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- what is geopandas?
- read and save geospatial data
- vector data processing

2 What is GeoPandas?

- GeoPandas, as the name suggests, extends the popular data science library `pandas` by adding support for geospatial data.
- GeoPandas builds on the capabilities of
  - Shapely: a Python package for the manipulation and analysis of geometric objects in the Cartesian plane (geoprocessing)
  - Pandas: a Python package that provides high-performance and easy-to-use data structures for data analysis in Python.
- Functionalities of Geopandas:
  - Read and Write geospatial data (vector)
  - Indexing and Selecting data
  - Geovisualization/Mapping
  - Manage projections
  - Geoprocessing: creating buffer, intersection between spatial objects, etc

2.1 Installation of GeoPandas

GeoPandas is written in pure Python, but has several **dependencies** written in C (GEOS, GDAL, PROJ). Those base C libraries can sometimes be a challenge to install. Geopandas developers advise users to closely follow the recommendations on their website to avoid installation problems.

From a terminal:

`pip install geopandas` you need to personally make sure that all dependencies are installed correctly - consult the geopandas website)

or

`conda install geopandas`

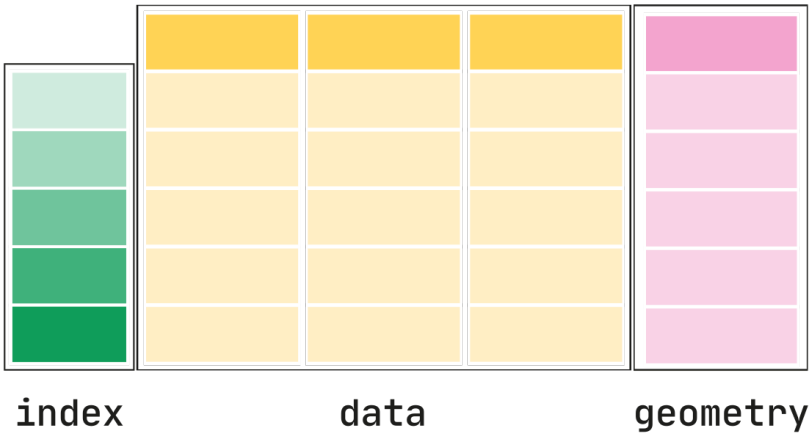
[1]:
```
import geopandas as gpd
```

[2]:
```
import pandas as pd
```

2.2 Core of GeoPandas: `GeoDataFrame`

- `geopandas.GeoDataFrame` is a subclass of `pandas.DataFrame`
  - The pandas `DataFrame` is a data structure that contains **two-dimensional** data and its corresponding row and column labels.
- `geopandas.GeoSeries` stores geometry columns and perform spatial operations.
  - `pandas.Series`: traditional data (numerical, boolean, text etc.)
  - `geopandas.GeoSeries`: a subclass of `pandas.Series` that handles the geometries
- In one instance of `geopandas.GeoDataFrame`, you can have as many columns with geometries as you wish; there's no limit typical for desktop GIS software

2.3 What is a Pandas `GeoDataFrame`?



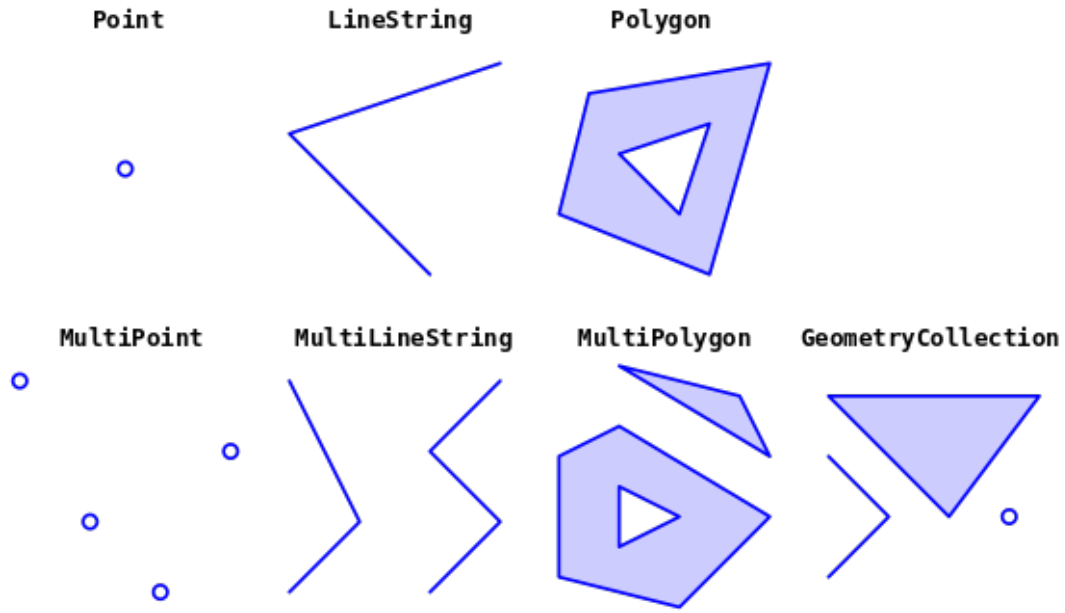
- gemetry is stored in one or more columns using the data structure `geopandas.GeoSeries`
  - A `GeoSeries` is essentially a vector where each entry in the vector is a set of shapes corresponding to one observation.
  - An entry may consist of
    - only one shape (like a single polygon) or

    - multiple shapes that are meant to be thought of as one observation (like the many polygons that make up the State of Hawaii or a country like Indonesia).
- traditional data are stored in other columns the same way as `pandas`

2.4 What geometries?

Three basic classes of geometric objects (which are actually `shapely` objects):

- Points / Multi-Points: e.g., trafic accident, house
- Lines / Multi-Lines: e.g., street
- Polygons / Multi-Polygons: e.g., census tract, county, state



2.5 Attributes and Methods of a geometric object/`GeoSeries`

- Attributes:
  - area: shape area
  - bounds: tuple of max and min coordinates on each axis for each shape
  - total_bounds: tuple of max and min coordinates on each axis for entire GeoSeries
  - geom_type: type of geometry.
- Methods:
  - distance(): returns minimum distance from each entry to other
  - centroid: returns GeoSeries of centroids
  - representative_point(): returns GeoSeries of points that are guaranteed to be within each geometry. It does NOT return centroids.
  - to_crs(): change coordinate reference system
  - plot(): mapping
  - contains(): is shape contained within other
  - intersects(): does shape intersect other

2.5.1 GeoSeries: Putting the Geo in GeoPandas

- We will create a few `shapely` Polygons/Points
- We are going to combine these three polygons in a geopandas `GeoSeries`

[3]:
```
from shapely.geometry import Polygon, Point
```

[4]:
```
p_1 = Point((0,0))
```

[5]:
```
type(p_1)
```

[5]:
```
shapely.geometry.point.Point
```

[6]:
```
poly_1 = Polygon([ (0,0), (0,10), (10, 10), (10, 0) ] )
poly_2 = Polygon([ (10,0), (10,10), (20, 10), (20, 0) ] )
poly_3 = Polygon([ (20,0), (20,10), (30, 10), (30, 0) ] )
```

[7]:
```
type(poly_1)
```

[7]:
```
shapely.geometry.polygon.Polygon
```

[8]:
```
polys1 = gpd.GeoSeries([p_1, poly_2, poly_3])
```

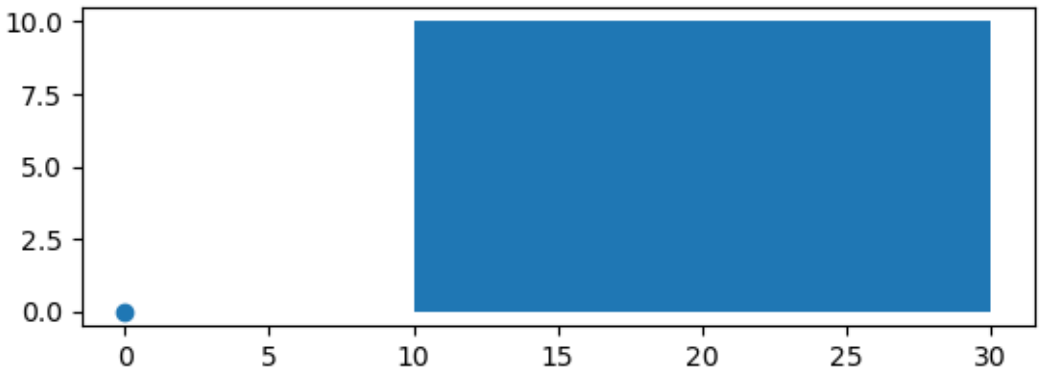
[9]:
```
type(polys1)
```

[9]:
```
geopandas.geoseries.GeoSeries
```

A `GeoSeries` can contain different types of geometries

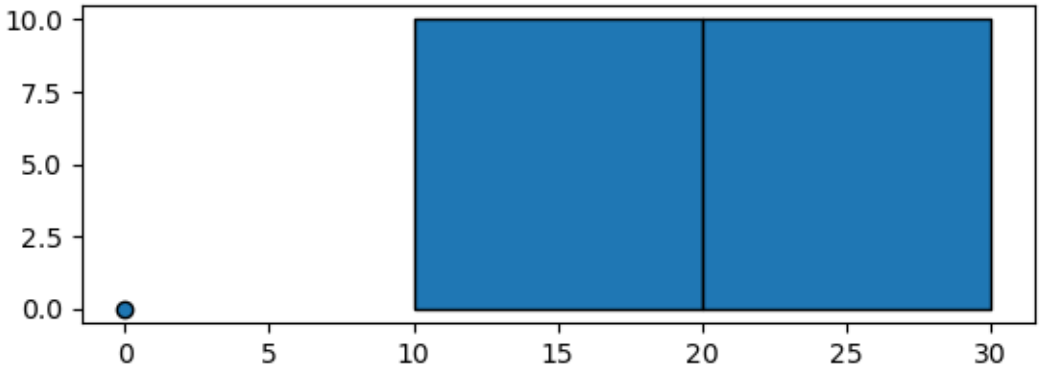
[10]:
```
polys1.plot()
```

[10]:
```
<Axes: >
```



[11]:
```
polys1.plot(edgecolor='k')
```

[11]: <Axes: >



The `GeoSeries` can be thought of as a vector, with each element of the vector corresponding to one or more Shapely geometry objects

[12]:
```
polys1
```

[12]:
```
0                              POINT (0.00000 0.00000)
1    POLYGON ((10.00000 0.00000, 10.00000 10.00000,…
2    POLYGON ((20.00000 0.00000, 20.00000 10.00000,…
dtype: geometry
```

[13]:
```
type(polys1[0])
```

[13]:
```
shapely.geometry.point.Point
```

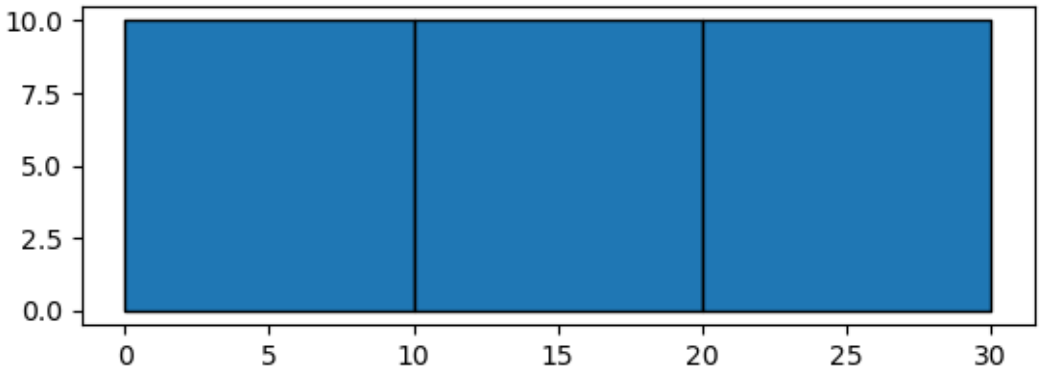
[14]:
```
type(polys1[1])
```

[14]:
```
shapely.geometry.polygon.Polygon
```

Create another `GeoSeries` with three polygons:

[15]:
```
polys2 = gpd.GeoSeries([poly_1, poly_2, poly_3])
polys2.plot(edgecolor='k')
```

[15]:
```
<Axes: >
```

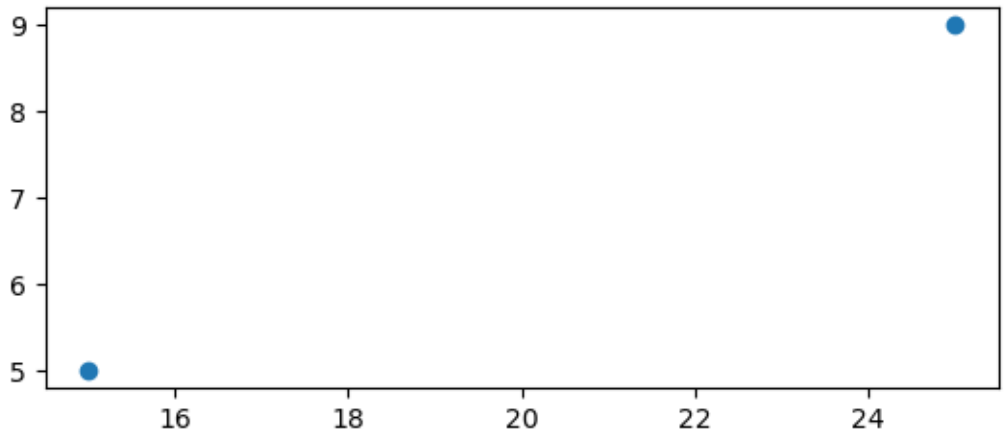


Determine spatial relationships: elementwise operations

[16]:
```
p_1 = Point(15, 5)
p_2 = Point(25, 9)
```

[17]:
```
points = gpd.GeoSeries([p_1, p_2])
points.plot()
```

[17]:
```
<Axes: >
```



[18]:
```
polys2.contains(p_1)
```

[18]:
```
0    False
1     True
2    False
dtype: bool
```

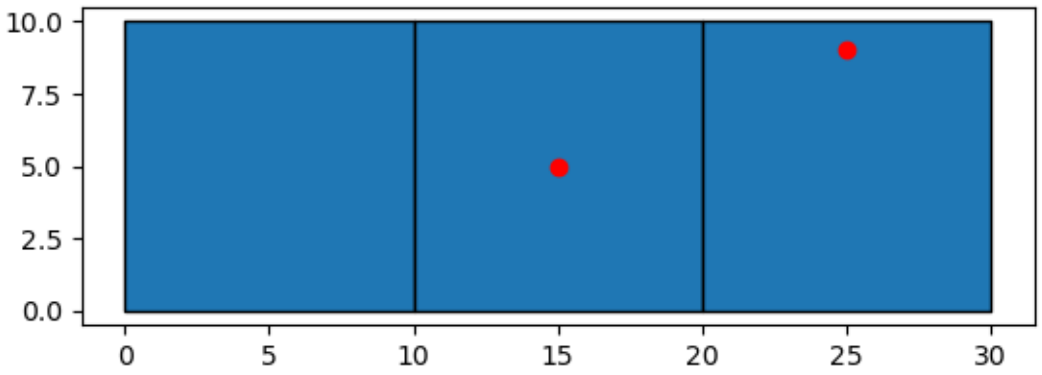
[19]:
```
polys2.contains(p_2)
```

[19]:
```
0    False
1    False
2     True
dtype: bool
```

We can plot the two `GeoSeries` to confirms their relationships:

[20]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
polys2.plot(ax=ax, edgecolor='k')
points.plot(ax=ax, edgecolor='r', facecolor='r')
```

[20]:
```
<Axes: >
```



2.6 GeoDataFrame

- `GeoSeries`: geometries
  - one or many
  - an active (default) geometry column
- `Series`: traditional columns

[21]:
```
gdf = gpd.GeoDataFrame({'region': ['west', 'central', 'east'], 'geometry':␣
↪polys2})
gdf
```

[21]:

	region	geometry
0	west	POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…
1	central	POLYGON ((10.00000 0.00000, 10.00000 10.00000,…
2	east	POLYGON ((20.00000 0.00000, 20.00000 10.00000,…

[22]:
```
gdf[["region", "geometry"]]
```

[22]:
```
    region                                           geometry
0     west  POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…
1  central  POLYGON ((10.00000 0.00000, 10.00000 10.00000,…
2     east  POLYGON ((20.00000 0.00000, 20.00000 10.00000,…
```

[23]:
```
gdf.region
```

[23]:
```
0       west
1    central
2       east
Name: region, dtype: object
```

add additional columns - identical to pandas

[24]:
```
gdf['Unemployment'] = [ 7.8, 5.3, 8.2]
gdf
```

[24]:
```
    region                                           geometry  Unemployment
0     west  POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…           7.8
1  central  POLYGON ((10.00000 0.00000, 10.00000 10.00000,…           5.3
2     east  POLYGON ((20.00000 0.00000, 20.00000 10.00000,…           8.2
```

Inherit all the functionalities of pandas when it comes to tradtional columns.

- supports different types of subsetting and traditional (i.e., nonspatial) queries.
- For example, find the regions with unemployment rates above 6 percent:

[25]:
```
gdf[gdf["Unemployment"] > 6]
```

[25]:
```
  region                                           geometry  Unemployment
0   west  POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…           7.8
2   east  POLYGON ((20.00000 0.00000, 20.00000 10.00000,…           8.2
```

We can add additional GeoSeries to the same dataframe, as they will be treated as regular columns.

However, only one GeoSeries can serve as the column against which any spatial methods are applied when called upon.

This column can be accessed through the `geometry` attribute of the `GeoDataFrame`:

[26]:
```
gdf
```

[26]:
```
    region                                           geometry  Unemployment
0     west  POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…           7.8
1  central  POLYGON ((10.00000 0.00000, 10.00000 10.00000,…           5.3
2     east  POLYGON ((20.00000 0.00000, 20.00000 10.00000,…           8.2
```

[27]:
```
gdf["geometry"]
```

[27]:
```
0    POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…
1    POLYGON ((10.00000 0.00000, 10.00000 10.00000,…
```

```
2    POLYGON ((20.00000 0.00000, 20.00000 10.00000,…
Name: geometry, dtype: geometry
```

[28]:
```
gdf.geometry
```

[28]:
```
0    POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…
1    POLYGON ((10.00000 0.00000, 10.00000 10.00000,…
2    POLYGON ((20.00000 0.00000, 20.00000 10.00000,…
Name: geometry, dtype: geometry
```

Let's create a new Points GeoSeries and add it to this GeoDataFrame as a regular column:

[29]:
```
points = gpd.GeoSeries([Point(5,5), Point(15, 6), Point([25,9])])
gdf['points'] = points
```

[30]:
```
gdf
```

[30]:
```
    region                                           geometry  Unemployment  \
0     west  POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…           7.8
1  central  POLYGON ((10.00000 0.00000, 10.00000 10.00000,…           5.3
2     east  POLYGON ((20.00000 0.00000, 20.00000 10.00000,…           8.2

                     points
0   POINT (5.00000 5.00000)
1  POINT (15.00000 6.00000)
2  POINT (25.00000 9.00000)
```

[31]:
```
gdf.geometry
```

[31]:
```
0    POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…
1    POLYGON ((10.00000 0.00000, 10.00000 10.00000,…
2    POLYGON ((20.00000 0.00000, 20.00000 10.00000,…
Name: geometry, dtype: geometry
```

So the `POLYGON` column is currently serving as the `geometry` property for the `GeoDataFrame` and `points` is just another "regular" column:

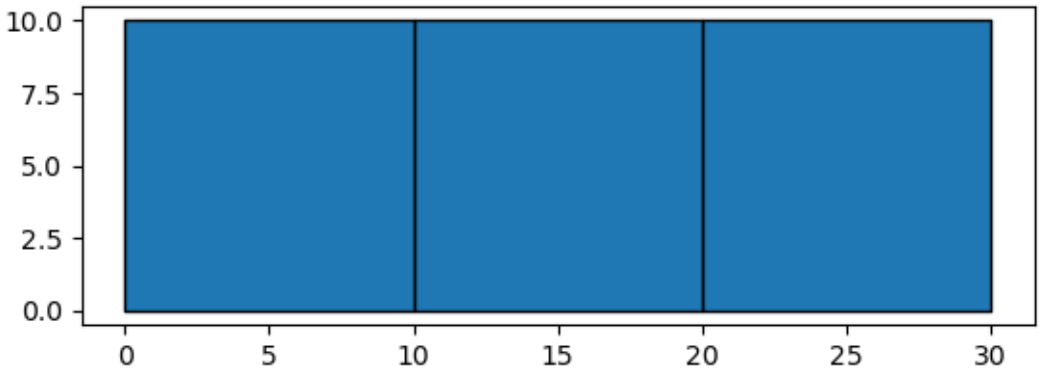
[32]:
```
gdf
```

[32]:
```
    region                                           geometry  Unemployment  \
0     west  POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…           7.8
1  central  POLYGON ((10.00000 0.00000, 10.00000 10.00000,…           5.3
2     east  POLYGON ((20.00000 0.00000, 20.00000 10.00000,…           8.2

                     points
0   POINT (5.00000 5.00000)
1  POINT (15.00000 6.00000)
2  POINT (25.00000 9.00000)
```

so when we call the `plot` method we get the polygon representation:

[33]:
```
gdf.plot(edgecolor='k')
```

[33]:
```
<Axes: >
```



We can explicity set the geometry property:

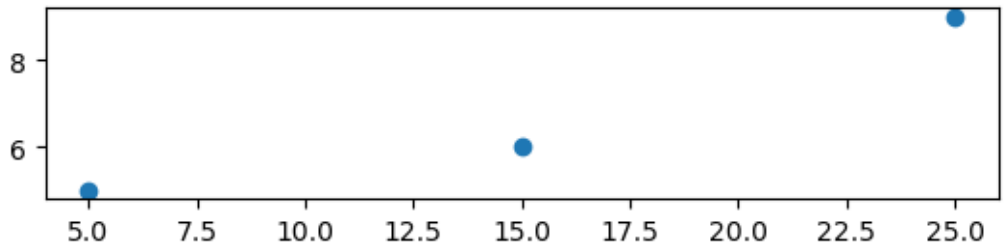
[34]:
```
gdf = gdf.set_geometry('points')
```

[35]:
```
gdf
```

[35]:
```
    region                                           geometry  Unemployment  \
0     west  POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…           7.8
1  central  POLYGON ((10.00000 0.00000, 10.00000 10.00000,…           5.3
2     east  POLYGON ((20.00000 0.00000, 20.00000 10.00000,…           8.2

                     points
0   POINT (5.00000 5.00000)
1  POINT (15.00000 6.00000)
2  POINT (25.00000 9.00000)
```

[36]:
```
gdf.plot()
```

[36]:
```
<Axes: >
```



[37]:
```
gdf
```

[37]:
```
    region                                           geometry  Unemployment  \
0     west  POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…           7.8
1  central  POLYGON ((10.00000 0.00000, 10.00000 10.00000,…           5.3
2     east  POLYGON ((20.00000 0.00000, 20.00000 10.00000,…           8.2

                     points
0   POINT (5.00000 5.00000)
1  POINT (15.00000 6.00000)
2  POINT (25.00000 9.00000)
```

[38]:
```
gdf.points
```

[38]:
```
0     POINT (5.00000 5.00000)
1    POINT (15.00000 6.00000)
2    POINT (25.00000 9.00000)
Name: points, dtype: geometry
```

[39]:
```
gdf["points"]
```

[39]:
```
0     POINT (5.00000 5.00000)
1    POINT (15.00000 6.00000)
2    POINT (25.00000 9.00000)
Name: points, dtype: geometry
```

[40]:
```
gdf["geometry"]
```

[40]:
```
0    POLYGON ((0.00000 0.00000, 0.00000 10.00000, 1…
1    POLYGON ((10.00000 0.00000, 10.00000 10.00000,…
2    POLYGON ((20.00000 0.00000, 20.00000 10.00000,…
Name: geometry, dtype: geometry
```

[41]:
```
gdf.geometry
```

[41]:
```
0     POINT (5.00000 5.00000)
1    POINT (15.00000 6.00000)
2    POINT (25.00000 9.00000)
Name: points, dtype: geometry
```

[42]:
```
gdf.geom_type # geometry type names
```

[42]:
```
0    Point
1    Point
2    Point
```

```
dtype: object
```

2.7 Read vector layers

`gpd.read_file` function can read vector layer files in any of the common formats, such as:

- Shapefile (.shp)
- GeoJSON (.geojson)
- GeoPackage (.gpkg)
- File Geodatabase (.gdb)

Read a polygon shapefile (census tracts in the Dallas–Fort Worth–Arlington metroplitan area)

[43]:
```
gdf = gpd.read_file("data/CensusTract_DallasMSA.shp")
```

[44]:
```
gdf.head(2)
```

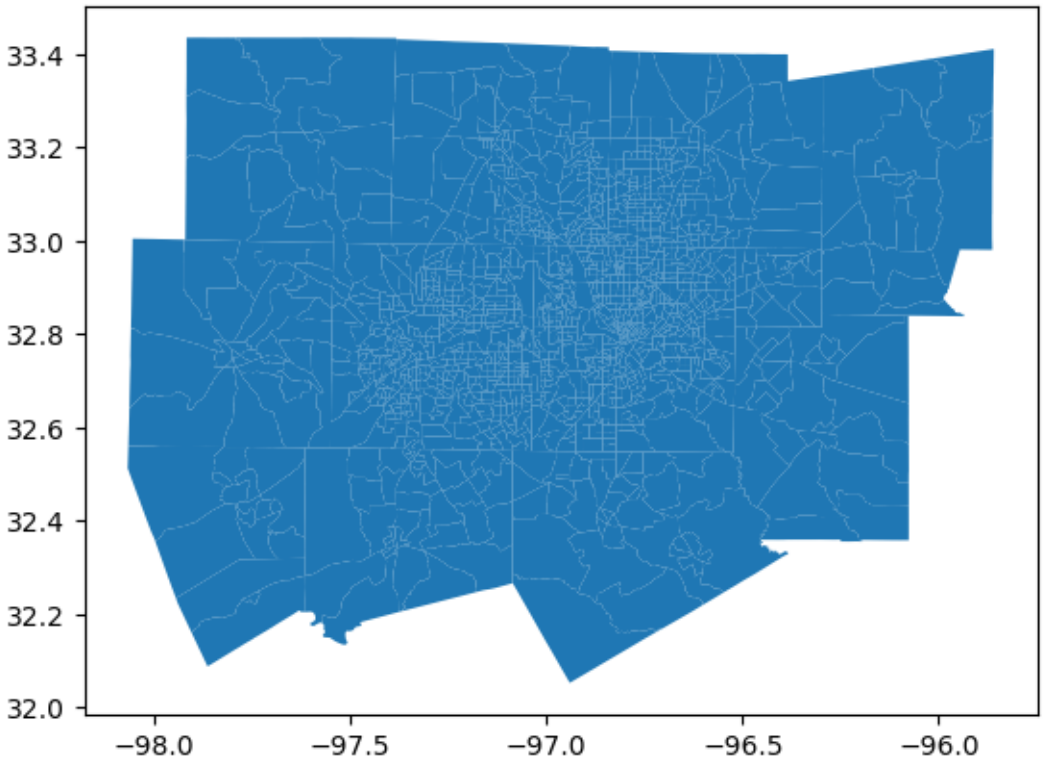
[44]:
```
  STATEFP COUNTYFP TRACTCE        GEOID    NAME             NAMELSAD  MTFCC  \
0      48      113  015900  48113015900     159     Census Tract 159  G5020
1      48      113  012604  48113012604  126.04  Census Tract 126.04  G5020

  FUNCSTAT    ALAND   AWATER     INTPTLAT      INTPTLON  \
0        S  5703840  4003497  +32.7362108  -096.9692130
1        S  2510513    43527  +32.8428141  -096.6460224

                                              geometry
0  POLYGON ((-96.99498 32.73739, -96.99492 32.737…
1  POLYGON ((-96.65925 32.84716, -96.65785 32.848…
```

[45]:
```
gdf.shape
```

[45]:
```
(1721, 13)
```

[46]:
```
gdf.plot()
```

[46]:
```
<Axes: >
```



[47]:
```
gdf.crs
```

[47]:
```
<Geographic 2D CRS: EPSG:4269>
Name: NAD83
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: North America - onshore and offshore: Canada - Alberta; British
Columbia; Manitoba; New Brunswick; Newfoundland and Labrador; Northwest
Territories; Nova Scotia; Nunavut; Ontario; Prince Edward Island; Quebec;
Saskatchewan; Yukon. Puerto Rico. United States (USA) - Alabama; Alaska;
Arizona; Arkansas; California; Colorado; Connecticut; Delaware; Florida;
Georgia; Hawaii; Idaho; Illinois; Indiana; Iowa; Kansas; Kentucky; Louisiana;
Maine; Maryland; Massachusetts; Michigan; Minnesota; Mississippi; Missouri;
Montana; Nebraska; Nevada; New Hampshire; New Jersey; New Mexico; New York;
North Carolina; North Dakota; Ohio; Oklahoma; Oregon; Pennsylvania; Rhode
Island; South Carolina; South Dakota; Tennessee; Texas; Utah; Vermont; Virginia;
Washington; West Virginia; Wisconsin; Wyoming. US Virgin Islands. British Virgin
Islands.
- bounds: (167.65, 14.92, -40.73, 86.45)
Datum: North American Datum 1983
```

```
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

[48]:
```
gdf.columns
```

[48]:
```
Index(['STATEFP', 'COUNTYFP', 'TRACTCE', 'GEOID', 'NAME', 'NAMELSAD', 'MTFCC',
       'FUNCSTAT', 'ALAND', 'AWATER', 'INTPTLAT', 'INTPTLON', 'geometry'],
      dtype='object')
```

[49]:
```
gdf.geom_type
```

[49]:
```
0       Polygon
1       Polygon
2       Polygon
3       Polygon
4       Polygon
         …
1716    Polygon
1717    Polygon
1718    Polygon
1719    Polygon
1720    Polygon
Length: 1721, dtype: object
```

Read a point shapefile (hospitals in the Dallas–Fort Worth–Arlington metroplitan area)

[50]:
```
gdf_hospitals = gpd.read_file("data/hospitals_dallasMSA.shp")
```

[51]:
```
gdf_hospitals.head()
```

[51]:
```
   OBJECTID          ID                                          NAME  \
0       237  0019676104               COOK CHILDREN'S MEDICAL CENTER
1       580  0004176015                    MEDICAL CITY OF ARLINGTON
2       581  0004276020  TEXAS HEALTH HARRIS METHODIST HOSPITAL AZLE
3       597  0021276104       BAYLOR SURGICAL HOSPITAL AT FORT WORTH
4       598  0021475034  BAYLOR SCOTT & WHITE MEDICAL CENTER - FRISCO

               ADDRESS        CITY STATE    ZIP           ZIP4  \
0   801 SEVENTH AVENUE  FORT WORTH    TX  76104  NOT AVAILABLE
1    3301 MATLOCK ROAD   ARLINGTON    TX  76015  NOT AVAILABLE
2     108 DENVER TRAIL        AZLE    TX  76020  NOT AVAILABLE
3      750 12TH AVENUE  FORT WORTH    TX  76104  NOT AVAILABLE
4  5601 WARREN PARKWAY      FRISCO    TX  75034  NOT AVAILABLE

        TELEPHONE               TYPE  …  \
0  (682) 885-4340           CHILDREN  …
1  (817) 465-3241  GENERAL ACUTE CARE  …
2  (817) 444-8600  GENERAL ACUTE CARE  …
```

```
3  (682) 703-5600  GENERAL ACUTE CARE  …
4  (214) 407-5000  GENERAL ACUTE CARE  …

                                          WEBSITE       STATE_ID  \
0                    http://www.cookchildrens.org  NOT AVAILABLE
1             http://www.medicalcenterarlington.com  NOT AVAILABLE
2                  http://www.texashealth.org/azle  NOT AVAILABLE
3                               http://bshfw.com/  NOT AVAILABLE
4  http://www.baylorhealth.com/physicianslocation…  NOT AVAILABLE

        ALT_NAME ST_FIPS        OWNER  TTL_STAFF   BEDS         TRAUMA  \
0  NOT AVAILABLE      48   NON-PROFIT       -999  430.0       LEVEL II
1  NOT AVAILABLE      48  PROPRIETARY       -999  342.0      LEVEL III
2  NOT AVAILABLE      48   NON-PROFIT       -999   36.0       LEVEL IV
3  NOT AVAILABLE      48  PROPRIETARY       -999   34.0  NOT AVAILABLE
4  NOT AVAILABLE      48  PROPRIETARY       -999   68.0  NOT AVAILABLE

  HELIPAD                    geometry
0       Y  POINT (-97.34141 32.73707)
1       Y  POINT (-97.11272 32.69233)
2       Y  POINT (-97.53310 32.88097)
3       N  POINT (-97.34898 32.73686)
4       N  POINT (-96.83783 33.10449)

[5 rows x 33 columns]
```

[52]:
```
gdf_hospitals.shape
```

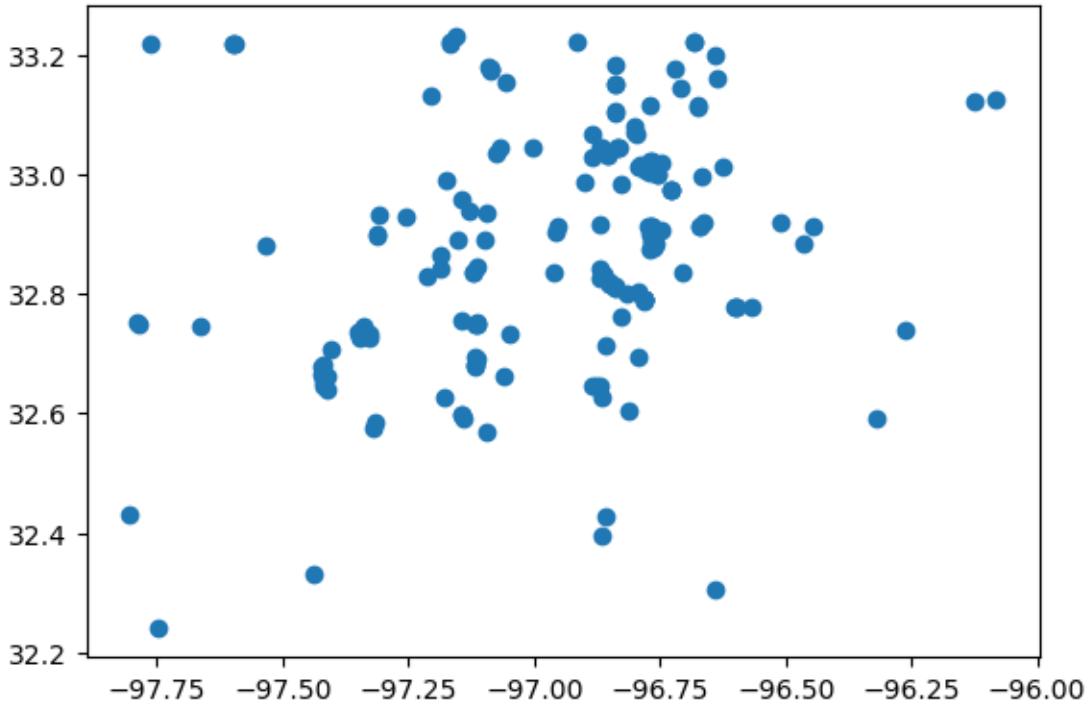
[52]:
```
(184, 33)
```

[53]:
```
gdf_hospitals.geom_type
```

[53]:
```
0      Point
1      Point
2      Point
3      Point
4      Point
       …
179    Point
180    Point
181    Point
182    Point
183    Point
Length: 184, dtype: object
```

[54]:
```
gdf_hospitals.plot()
```

[54]: <Axes: >



[55]:
```
gdf_hospitals.crs
```

[55]:
```
<Geographic 2D CRS: EPSG:4269>
Name: NAD83
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: North America - onshore and offshore: Canada - Alberta; British
Columbia; Manitoba; New Brunswick; Newfoundland and Labrador; Northwest
Territories; Nova Scotia; Nunavut; Ontario; Prince Edward Island; Quebec;
Saskatchewan; Yukon. Puerto Rico. United States (USA) - Alabama; Alaska;
Arizona; Arkansas; California; Colorado; Connecticut; Delaware; Florida;
Georgia; Hawaii; Idaho; Illinois; Indiana; Iowa; Kansas; Kentucky; Louisiana;
Maine; Maryland; Massachusetts; Michigan; Minnesota; Mississippi; Missouri;
Montana; Nebraska; Nevada; New Hampshire; New Jersey; New Mexico; New York;
North Carolina; North Dakota; Ohio; Oklahoma; Oregon; Pennsylvania; Rhode
Island; South Carolina; South Dakota; Tennessee; Texas; Utah; Vermont; Virginia;
Washington; West Virginia; Wisconsin; Wyoming. US Virgin Islands. British Virgin
Islands.
- bounds: (167.65, 14.92, -40.73, 86.45)
Datum: North American Datum 1983
```

```
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

2.7.1 Spatial join

Spatial Join: two geometry objects are merged based on their spatial relationship to one another.

`GeoDataFrame.sjoin()`

- keyword argument `Predicate`: specifies how geopandas decides whether or not to join the attributes of one object to another, based on their geometric relationship.
  - `"intersects"` (default): the boundary or interior of the object intersect in any way with those of the other.
  - `"contains"`
  - `"within"`: object's boundary and interior intersect only with the interior of the other (not its boundary or exterior).
  - `"touches"`
  - `"crosses"`: the interior of the object intersects the interior of the other but does not contain it, and the dimension of the intersection is less than the dimension of the one or the other.
  - `"overlaps"`
- keyword argument `how`: The type of join
  - `"inner"` (default): use intersection of keys from both dfs; retain only left_df geometry column
  - `"left"`: use keys from left_df; retain only left_df geometry column
  - `"right"`: use keys from right_df; retain only right_df geometry column

Read more here on the specific meaning of each of the predicates. More on spatial join with GeoPandas

We will join the point data with the polygon data based on their spatial relationships: a point (hospital) is within a a polygon (census tract). Therefore, we use the predicate `within` and put the point geodataframe before the polygon geodataframe.

[56]:
```
gdf_hospitals  = gdf_hospitals.to_crs(gdf.crs) #making sure these the two␣
↪geometries have the same projection system
```

[57]:
```
hospital_tracts = gpd.sjoin(gdf_hospitals, gdf, predicate='within')
```

[58]:
```
hospital_tracts.shape
```

[58]:
```
(184, 46)
```

[59]:
```
gdf_hospitals.shape
```

[59]:
```
(184, 33)
```

[60]:
```
gdf.shape
```

[60]:
```
(1721, 13)
```

[61]:
```
hospital_tracts.columns
```

[61]:
```
Index(['OBJECTID', 'ID', 'NAME_left', 'ADDRESS', 'CITY', 'STATE', 'ZIP',
       'ZIP4', 'TELEPHONE', 'TYPE', 'STATUS', 'POPULATION', 'COUNTY',
       'COUNTYFIPS', 'COUNTRY', 'LATITUDE', 'LONGITUDE', 'NAICS_CODE',
       'NAICS_DESC', 'SOURCE', 'SOURCEDATE', 'VAL_METHOD', 'VAL_DATE',
       'WEBSITE', 'STATE_ID', 'ALT_NAME', 'ST_FIPS', 'OWNER', 'TTL_STAFF',
       'BEDS', 'TRAUMA', 'HELIPAD', 'geometry', 'index_right', 'STATEFP',
       'COUNTYFP', 'TRACTCE', 'GEOID', 'NAME_right', 'NAMELSAD', 'MTFCC',
       'FUNCSTAT', 'ALAND', 'AWATER', 'INTPTLAT', 'INTPTLON'],
      dtype='object')
```

[62]:
```
hospital_tracts.geometry
```

[62]:
```
0      POINT (-97.34141 32.73707)
3      POINT (-97.34898 32.73686)
33     POINT (-97.33900 32.73676)
35     POINT (-97.34554 32.73022)
53     POINT (-97.34335 32.73577)
                  …
173    POINT (-97.20473 33.13214)
175    POINT (-96.83890 33.18138)
179    POINT (-96.88528 33.06618)
180    POINT (-96.80180 33.07079)
182    POINT (-97.66231 32.74480)
Name: geometry, Length: 184, dtype: geometry
```

The active geometry of the GeoDataFrame after spatial join (default setting) will be the geometry of the first GeoDataFrame. The geometry from the second GeoDataFrame will be abandoned.

2.7.2 Aggregation

Calculate the number of hospitals in each tract

More on Group by: split-apply-combine

[63]:
```
hospital_tracts.head()
```

[63]:

	OBJECTID	ID	NAME_left	ADDRESS	CITY	STATE	ZIP	ZIP4
0	237	0019676104	COOK CHILDREN'S MEDICAL CENTER	801 SEVENTH AVENUE	FORT WORTH	TX	76104	NOT AVAILABLE
3	597	0021276104	BAYLOR SURGICAL HOSPITAL AT FORT WORTH	750 12TH AVENUE	FORT WORTH	TX	76104	NOT AVAILABLE
33	1395	0019976104	TEXAS HEALTH HARRIS METHODIST HOSPITAL FORT WORTH	1301 PENNSYLVANIA AVENUE	FORT WORTH	TX	76104	NOT AVAILABLE
35	1397	0020176101	BAYLOR SCOTT & WHITE ALL SAINTS MEDICAL CENTER…					
53	1544	0021076104	KINDRED HOSPITAL - FORT WORTH					

```
35          1400 8TH AVENUE  FORT WORTH    TX  76104  NOT AVAILABLE
53        815 EIGHTH AVENUE  FORT WORTH    TX  76104  NOT AVAILABLE

         TELEPHONE                TYPE  … TRACTCE        GEOID NAME_right  \
0   (682) 885-4340            CHILDREN  …  123700  48439123700       1237
3   (682) 703-5600  GENERAL ACUTE CARE  …  123700  48439123700       1237
33  (817) 250-2000  GENERAL ACUTE CARE  …  123700  48439123700       1237
35  (817) 926-2544  GENERAL ACUTE CARE  …  123700  48439123700       1237
53  (817) 332-4812       LONG TERM CARE  …  123700  48439123700       1237

              NAMELSAD  MTFCC  FUNCSTAT    ALAND AWATER     INTPTLAT  \
0   Census Tract 1237  G5020         S  3532535  48230  +32.7327574
3   Census Tract 1237  G5020         S  3532535  48230  +32.7327574
33  Census Tract 1237  G5020         S  3532535  48230  +32.7327574
35  Census Tract 1237  G5020         S  3532535  48230  +32.7327574
53  Census Tract 1237  G5020         S  3532535  48230  +32.7327574

        INTPTLON
0   -097.3447552
3   -097.3447552
33  -097.3447552
35  -097.3447552
53  -097.3447552

[5 rows x 46 columns]
```

[64]:
```
hospital_tracts.groupby('GEOID').size()
```

[64]:
```
GEOID
48085030301    2
48085030410    2
48085030504    2
48085030507    1
48085030519    1
              ..
48439123301    1
48439123600    2
48439123700    9
48497150201    3
48497150500    1
Length: 121, dtype: int64
```

Now that we have the number of hospitals in each tract, we can add this column to the original tract-level GeoDataFrame `gdf`.

[65]:
```
gdf = gdf.set_index("GEOID")
gdf.head()
```

[65]:
```
            STATEFP COUNTYFP TRACTCE    NAME             NAMELSAD  MTFCC  \
GEOID
48113015900      48      113  015900     159     Census Tract 159  G5020
48113012604      48      113  012604  126.04  Census Tract 126.04  G5020
48113013010      48      113  013010  130.10  Census Tract 130.10  G5020
48113013622      48      113  013622  136.22  Census Tract 136.22  G5020
48113013621      48      113  013621  136.21  Census Tract 136.21  G5020

            FUNCSTAT    ALAND   AWATER     INTPTLAT      INTPTLON  \
GEOID
48113015900        S  5703840  4003497  +32.7362108  -096.9692130
48113012604        S  2510513    43527  +32.8428141  -096.6460224
48113013010        S  1404466        0  +32.8688147  -096.6802329
48113013622        S  1249629        0  +32.9496147  -096.8110809
48113013621        S   910824      686  +32.9605446  -096.8087578

                                                    geometry
GEOID
48113015900  POLYGON ((-96.99498 32.73739, -96.99492 32.737…
48113012604  POLYGON ((-96.65925 32.84716, -96.65785 32.848…
48113013010  POLYGON ((-96.69384 32.87418, -96.69347 32.874…
48113013622  POLYGON ((-96.81880 32.95231, -96.81878 32.952…
48113013621  POLYGON ((-96.81198 32.95712, -96.81197 32.957…
```

[66]:
```
gdf["NumHosp"] = hospital_tracts.groupby('GEOID').size()
```

When adding the new columns, GeoPandas will align the records (rows) based on the index value (`GEOID`). Or we can use the `merge` method inherited from the `pandas.DataFrame`:

```
gdf.merge(hospital_tracts.groupby('GEOID').size(), left_index=True, right_index=True)
```

[67]:
```
gdf.head()
```

[67]:
```
            STATEFP COUNTYFP TRACTCE    NAME             NAMELSAD  MTFCC  \
GEOID
48113015900      48      113  015900     159     Census Tract 159  G5020
48113012604      48      113  012604  126.04  Census Tract 126.04  G5020
48113013010      48      113  013010  130.10  Census Tract 130.10  G5020
48113013622      48      113  013622  136.22  Census Tract 136.22  G5020
48113013621      48      113  013621  136.21  Census Tract 136.21  G5020

            FUNCSTAT    ALAND   AWATER     INTPTLAT      INTPTLON  \
GEOID
48113015900        S  5703840  4003497  +32.7362108  -096.9692130
48113012604        S  2510513    43527  +32.8428141  -096.6460224
48113013010        S  1404466        0  +32.8688147  -096.6802329
48113013622        S  1249629        0  +32.9496147  -096.8110809
48113013621        S   910824      686  +32.9605446  -096.8087578
```

```
                                                      geometry  NumHosp
GEOID
48113015900  POLYGON ((-96.99498 32.73739, -96.99492 32.737…       NaN
48113012604  POLYGON ((-96.65925 32.84716, -96.65785 32.848…       NaN
48113013010  POLYGON ((-96.69384 32.87418, -96.69347 32.874…       NaN
48113013622  POLYGON ((-96.81880 32.95231, -96.81878 32.952…       NaN
48113013621  POLYGON ((-96.81198 32.95712, -96.81197 32.957…       NaN
```

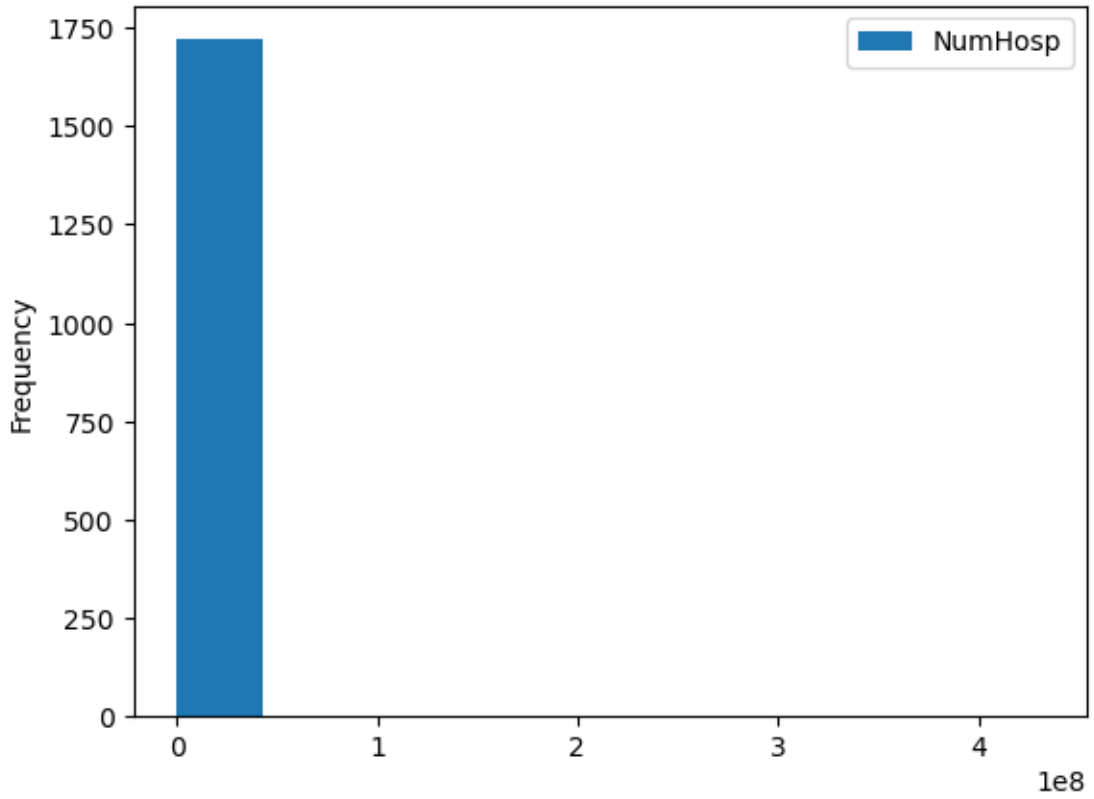
For census tracts where there is not a value from the number of hospital Series, missing value (nan) is used in the new column

[68]:
```
gdf.NumHosp = gdf['NumHosp'].fillna(0)
```

Use method `.fillna` to replace missing values with 0

[69]:
```
gdf.plot.hist(column=["NumHosp"])
```

[69]:
```
<Axes: ylabel='Frequency'>
```



[70]:
```
gdf.plot("NumHosp", legend=True)
```

[70]: <Axes: >

3 Writing Spatial Data

GeoDataFrames can be exported to many different standard formats using the `geopandas.GeoDataFrame.to_file()` method:

- Shapefile (.shp)
- GeoJSON (.geojson)
- GeoPackage (.gpkg)

For a full list of supported formats, type `import fiona; fiona.supported_drivers`

[71]:
```
gdf.to_file("data/CensusTract_DallasMSA_hospitals.shp")
```

4 Further readings

- Rey, S.J., D. Arribas-Bel, and L.J. Wolf, Geographical Data Science with Python.
- GeoPandas documentation
- Shapely documentation

[ ]: