15.1_Mapping

November 6, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- Cartography and geovisualization with geopandas

2 Cartography and Geovisualization

- Cartography: traditional methods of making and using maps to communicate spatial information effectively.
- Geovisualization, also known as cartographic visualization, refers to a set of tools and techniques supporting the analysis of geospatial data through the use of **interactive** visualization.

Two types:

- static maps
- interactive maps

2.1 Making static maps for presentation/publication

GeoPandas provides a high-level interface to the matplotlib library for making maps.

With GeoPandas, we can make thematic maps for presenting spatial patterns of a specific characteristic:

```
GeoDataFrame.plot(column  = [column name])
```

Also called choropleth maps (maps where the color of each shape is based on the value of an associated variable)

[1]:
```
import geopandas as gpd
import pandas as pd
import numpy as np
```

[3]:
```
gdf_tract = gpd.read_file("data/CensusTract_DallasMSA_hospitals.shp")
```

[4]:
```
gdf_tract.head()
```

[4]:

	GEOID	STATEFP	COUNTYFP	TRACTCE	NAME	NAMELSAD	MTFCC	\
0	48113015900	48	113	015900	159	Census Tract 159	G5020	
1	48113012604	48	113	012604	126.04	Census Tract 126.04	G5020	
2	48113013010	48	113	013010	130.10	Census Tract 130.10	G5020	
3	48113013622	48	113	013622	136.22	Census Tract 136.22	G5020	
4	48113013621	48	113	013621	136.21	Census Tract 136.21	G5020	

	FUNCSTAT	ALAND	AWATER	INTPTLAT	INTPTLON	NumHosp	\
0	S	5703840	4003497	+32.7362108	-096.9692130	0.0	
1	S	2510513	43527	+32.8428141	-096.6460224	0.0	
2	S	1404466	0	+32.8688147	-096.6802329	0.0	
3	S	1249629	0	+32.9496147	-096.8110809	0.0	
4	S	910824	686	+32.9605446	-096.8087578	0.0	

	geometry
0	POLYGON ((-96.99498 32.73739, -96.99492 32.737…
1	POLYGON ((-96.65925 32.84716, -96.65785 32.848…
2	POLYGON ((-96.69384 32.87418, -96.69347 32.874…
3	POLYGON ((-96.81880 32.95231, -96.81878 32.952…
4	POLYGON ((-96.81198 32.95712, -96.81197 32.957…

[6]:
```
df_tract = pd.read_csv("data/ACS2020_Dallas_Demographics.csv")
```

[7]:
```
df_tract.head()
```

[7]:

	GEOID	county	Pop	White	Black	Asian	Hispanic
0	48085031811	85	1662	57.76	2.29	24.19	11.43
1	48085031812	85	1120	83.04	0.00	12.23	4.73
2	48085031813	85	2287	41.41	8.48	25.54	22.82
3	48085031814	85	1932	72.20	1.97	2.69	22.05
4	48085031815	85	3764	75.58	6.93	8.48	4.97

Merge the census tract shapefile for the Dallas metro area with the table of tract-level demographics.

The merge/join will be based on the tract id: `GEOID` column in `gdf_tract` and `GEOID` column in `df_tract`.

[8]:
```
gdf_tract = gdf_tract.merge(df_tract, left_on = "GEOID",
                            right_on="GEOID")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[8], line 1
----> 1 gdf_tract = gdf_tract.merge(df_tract, left_on = "GEOID",
      2                            right_on="GEOID")

File ~/anaconda3/lib/python3.11/site-packages/geopandas/geodataframe.py:1574, i␣
 ↪GeoDataFrame.merge(self, *args, **kwargs)
   1556 def merge(self, *args, **kwargs):
   1557     r"""Merge two ``GeoDataFrame`` objects with a database-style join.
   1558
   1559     Returns a ``GeoDataFrame`` if a geometry column is present;␣
 ↪otherwise,
   (…)
   1572     for more details.
   1573     """
-> 1574     result = DataFrame.merge(self, *args, **kwargs)
   1575     geo_col = self._geometry_column_name
   1576     if isinstance(result, DataFrame) and geo_col in result:

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/frame.py:9843, in␣
 ↪DataFrame.merge(self, right, how, on, left_on, right_on, left_index,␣
 ↪right_index, sort, suffixes, copy, indicator, validate)
   9824 @Substitution("")
   9825 @Appender(_merge_doc, indents=2)
   9826 def merge(
   (…)
   9839     validate: str | None = None,
   9840 ) -> DataFrame:
   9841     from pandas.core.reshape.merge import merge
-> 9843     return merge(
   9844         self,
   9845         right,
   9846         how=how,
   9847         on=on,
   9848         left_on=left_on,
   9849         right_on=right_on,
   9850         left_index=left_index,
   9851         right_index=right_index,
   9852         sort=sort,
   9853         suffixes=suffixes,
   9854         copy=copy,
   9855         indicator=indicator,
   9856         validate=validate,
```

```
   9857     )

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/merge.py:148,
 ↪in merge(left, right, how, on, left_on, right_on, left_index, right_index,␣
 ↪sort, suffixes, copy, indicator, validate)
    131 @Substitution("\nleft : DataFrame or named Series")
    132 @Appender(_merge_doc, indents=0)
    133 def merge(
   (…)
    146     validate: str | None = None,
    147 ) -> DataFrame:
--> 148     op = _MergeOperation(
    149         left,
    150         right,
    151         how=how,
    152         on=on,
    153         left_on=left_on,
    154         right_on=right_on,
    155         left_index=left_index,
    156         right_index=right_index,
    157         sort=sort,
    158         suffixes=suffixes,
    159         indicator=indicator,
    160         validate=validate,
    161     )
    162     return op.get_result(copy=copy)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/merge.py:741,
 ↪in _MergeOperation.__init__(self, left, right, how, on, left_on, right_on,␣
 ↪axis, left_index, right_index, sort, suffixes, indicator, validate)
    733 (
    734     self.left_join_keys,
    735     self.right_join_keys,
    736     self.join_names,
    737 ) = self._get_merge_keys()
    739 # validate the merge keys dtypes. We may need to coerce
    740 # to avoid incompatible dtypes
--> 741 self._maybe_coerce_merge_keys()
    743 # If argument passed to validate,
    744 # check if columns specified as unique
    745 # are in fact unique.
    746 if validate is not None:

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/merge.py:1401
 ↪in _MergeOperation._maybe_coerce_merge_keys(self)
   1395     # unless we are merging non-string-like with string-like
   1396     elif (
```

```
  1397         inferred_left in string_types and inferred_right not in␣
↪string_types
  1398     ) or (
  1399         inferred_right in string_types and inferred_left not in␣
↪string_types
  1400     ):
-> 1401         raise ValueError(msg)
  1403 # datetimelikes must match exactly
  1404 elif needs_i8_conversion(lk.dtype) and not needs_i8_conversion(rk.dtype :

ValueError: You are trying to merge on object and int64 columns. If you wish to␣
↪proceed you should use pd.concat
```

Errors due to mismatched data types in the `GEOID` columns.

- In the GeoDataFrame `gdf_tract`, `GEOID` column has the type `object/str`
- In the DataFrame `df_tract`, `GEOID` column has the type `int`

[9]:
```
df_tract.GEOID.dtype
```

[9]:
```
dtype('int64')
```

[10]:
```
gdf_tract.GEOID.dtype
```

[10]:
```
dtype('O')
```

We need to change the type in one of them:

[11]:
```
df_tract["GEOID"] = df_tract["GEOID"].astype(str)
```

[12]:
```
df_tract.GEOID.dtype
```

[12]:
```
dtype('O')
```

[13]:
```
gdf_tract = gdf_tract.merge(df_tract, left_on = "GEOID", right_on="GEOID")
```

[14]:
```
gdf_tract.head()
```

[14]:

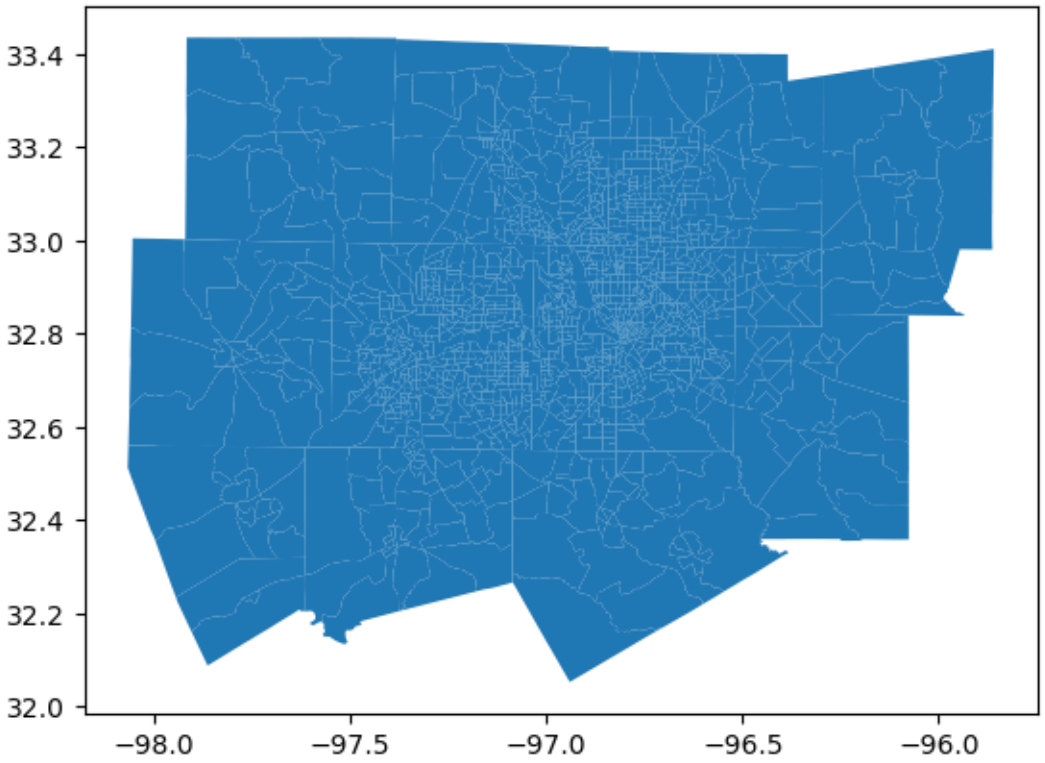
	GEOID	STATEFP	COUNTYFP	TRACTCE	NAME	NAMELSAD	MTFCC	\
0	48113015900	48	113	015900	159	Census Tract 159	G5020	
1	48113012604	48	113	012604	126.04	Census Tract 126.04	G5020	
2	48113013010	48	113	013010	130.10	Census Tract 130.10	G5020	
3	48113013622	48	113	013622	136.22	Census Tract 136.22	G5020	
4	48113013621	48	113	013621	136.21	Census Tract 136.21	G5020	

	FUNCSTAT	ALAND	AWATER	INTPTLAT	INTPTLON	NumHosp	\
0	S	5703840	4003497	+32.7362108	-096.9692130	0.0	
1	S	2510513	43527	+32.8428141	-096.6460224	0.0	
2	S	1404466	0	+32.8688147	-096.6802329	0.0	

```
3        S  1249629        0  +32.9496147  -096.8110809      0.0
4        S   910824      686  +32.9605446  -096.8087578      0.0

                                                geometry  county   Pop  White  \
0  POLYGON ((-96.99498 32.73739, -96.99492 32.737…     113  3611  18.55
1  POLYGON ((-96.65925 32.84716, -96.65785 32.848…     113  7387  21.10
2  POLYGON ((-96.69384 32.87418, -96.69347 32.874…     113  5149  17.58
3  POLYGON ((-96.81880 32.95231, -96.81878 32.952…     113  2811  51.19
4  POLYGON ((-96.81198 32.95712, -96.81197 32.957…     113  4462  35.30

   Black  Asian  Hispanic
0   7.59   1.99     68.73
1  29.48   1.81     46.87
2  32.88   2.23     47.15
3  16.47   2.77     25.15
4  28.91   3.72     26.80
```

[15]:
```
gdf_tract.plot() # Basic plot, random colors
```

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

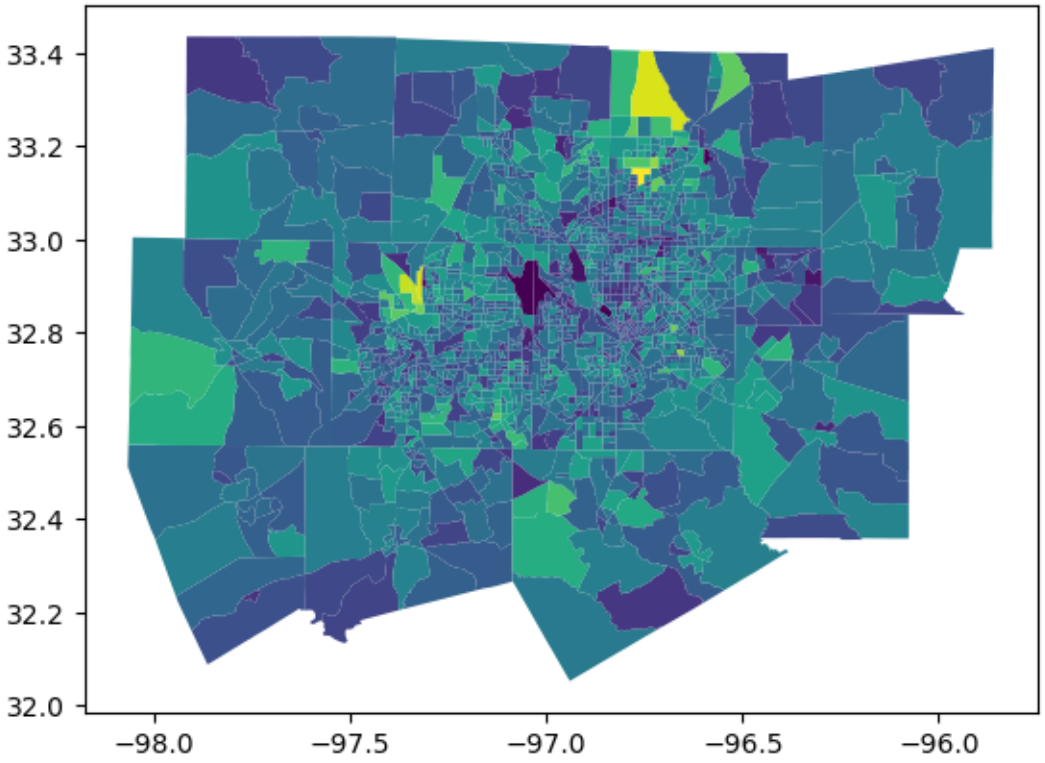


[16]:
```
gdf_tract.head(1)
```

[16]:
```
          GEOID STATEFP COUNTYFP TRACTCE NAME          NAMELSAD  MTFCC  \
0  48113015900      48      113  015900  159  Census Tract 159  G5020

  FUNCSTAT    ALAND   AWATER     INTPTLAT      INTPTLON  NumHosp  \
0        S  5703840  4003497  +32.7362108  -096.9692130      0.0

                                              geometry  county   Pop  White  \
0  POLYGON ((-96.99498 32.73739, -96.99492 32.737…     113  3611  18.55

   Black  Asian  Hispanic
0   7.59   1.99     68.73
```

[17]:
```
gdf_tract.plot("Pop")
```

[17]: <Axes: >



Creating a legend

[18]:
```
gdf_tract.plot("Pop",legend=True)
```

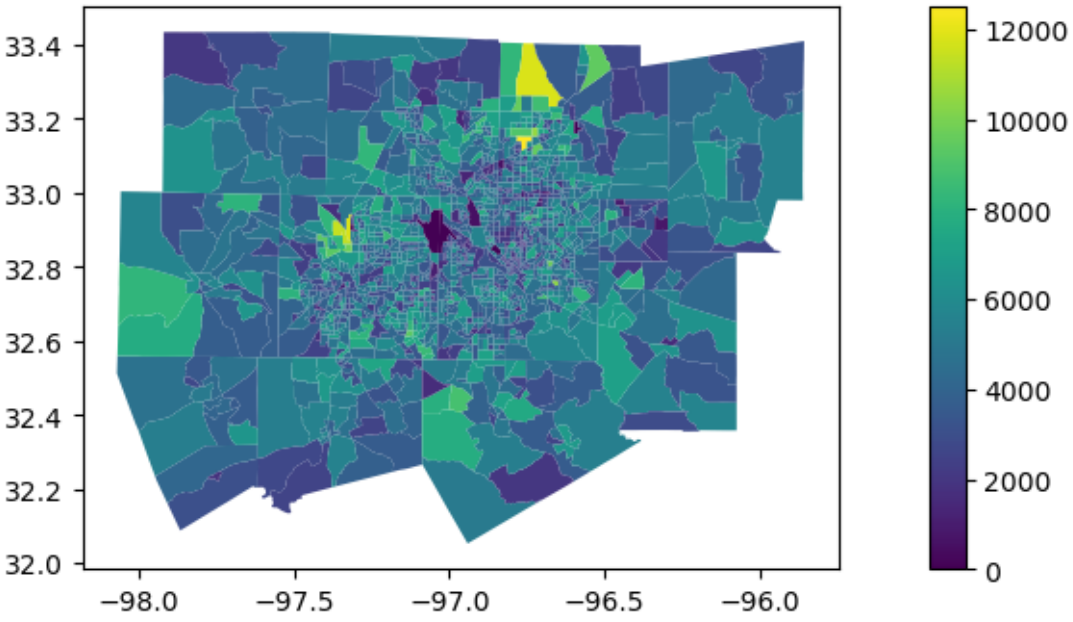
[18]: <Axes: >

We may want to change the default appearance of the legend and plot axes so that they align better.

We can define the plot axes (with `ax`) and the legend axes (with `cax`) and then pass those in to the `plot()` call.

[19]:
```
from mpl_toolkits.axes_grid1 import make_axes_locatable
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="4%", pad=0.1)
gdf_tract.plot("Pop",legend=True, ax=ax, cax=cax)
```

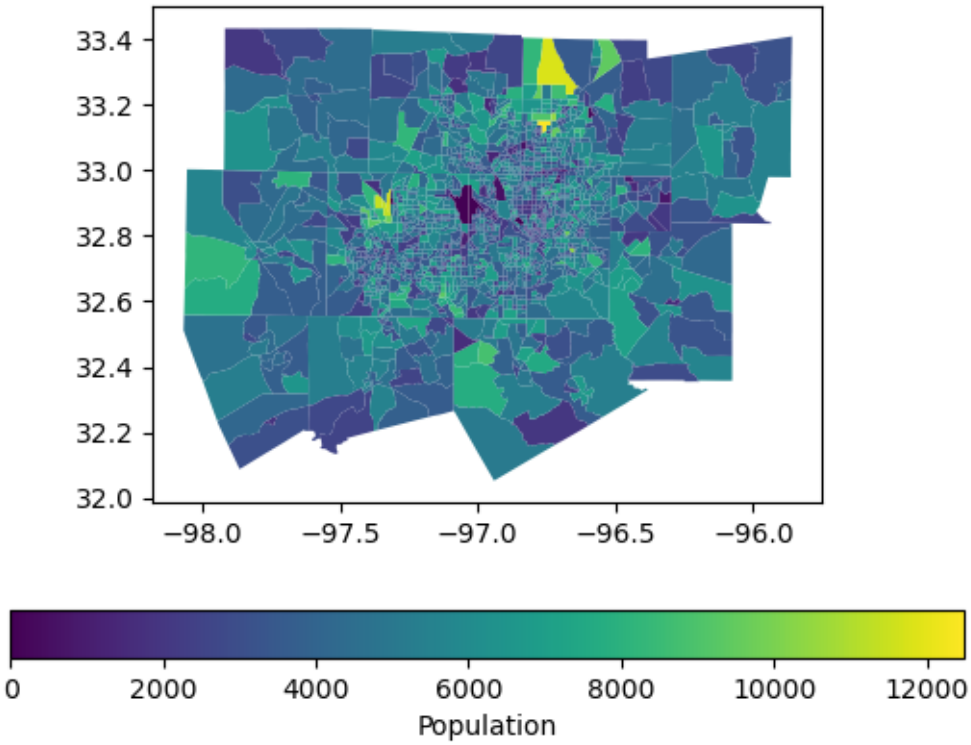
[19]: <Axes: >



we can also plot the color bar below the map

[20]:
```
fig, ax = plt.subplots()
gdf_tract.plot("Pop",legend=True, ax=ax,
               legend_kwds={'label': "Population",
                            'orientation': "horizontal"})
```

[20]: <Axes: >

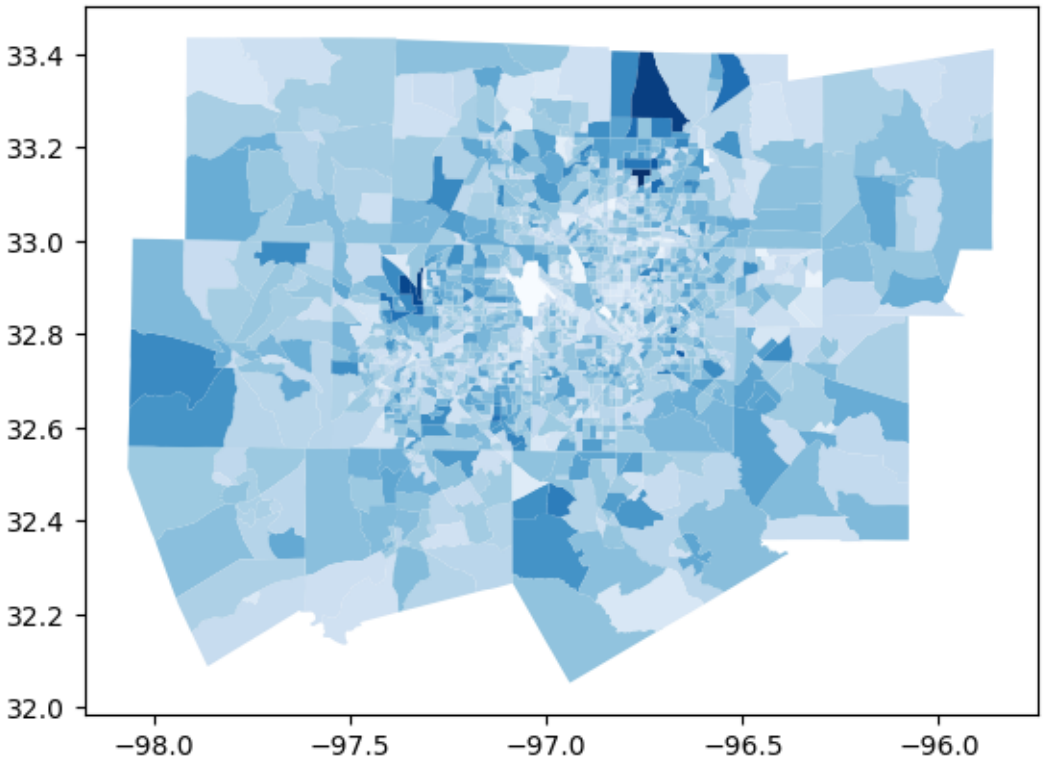


Changing colors for the maps by passing value to `cmap`

A list of available colormaps from matplotlib

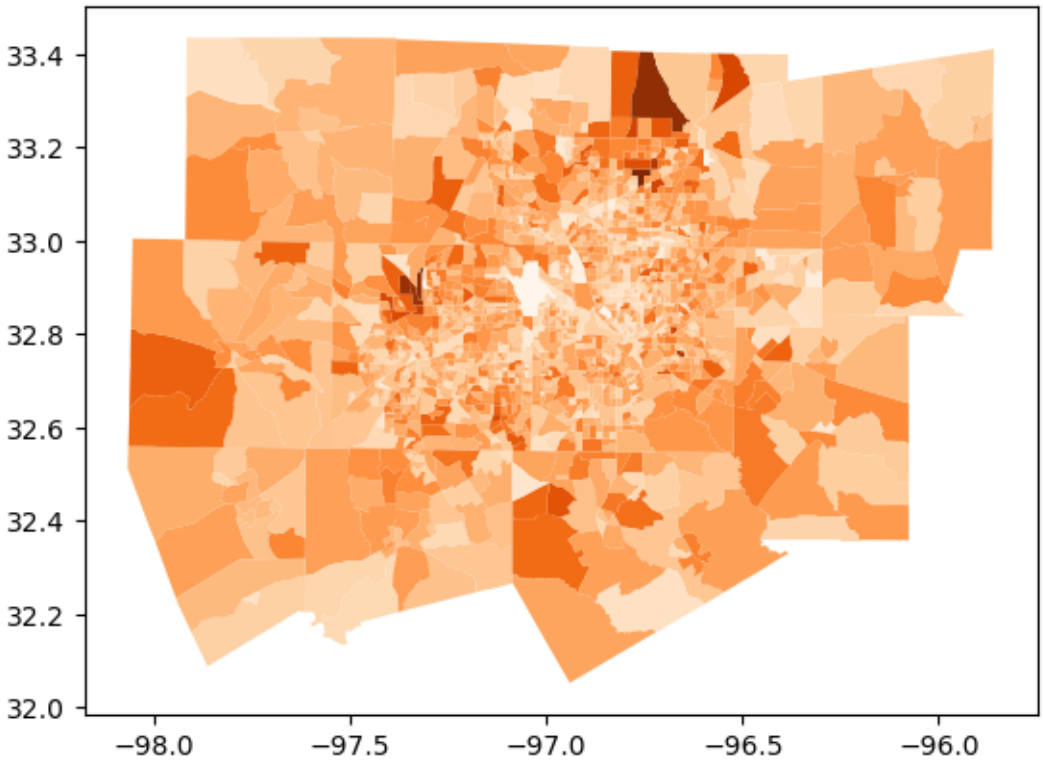
[21]:
```
gdf_tract.plot("Pop", cmap='Blues')
```

[21]: `<Axes: >`



[22]:
```
gdf_tract.plot("Pop", cmap='Oranges')
```

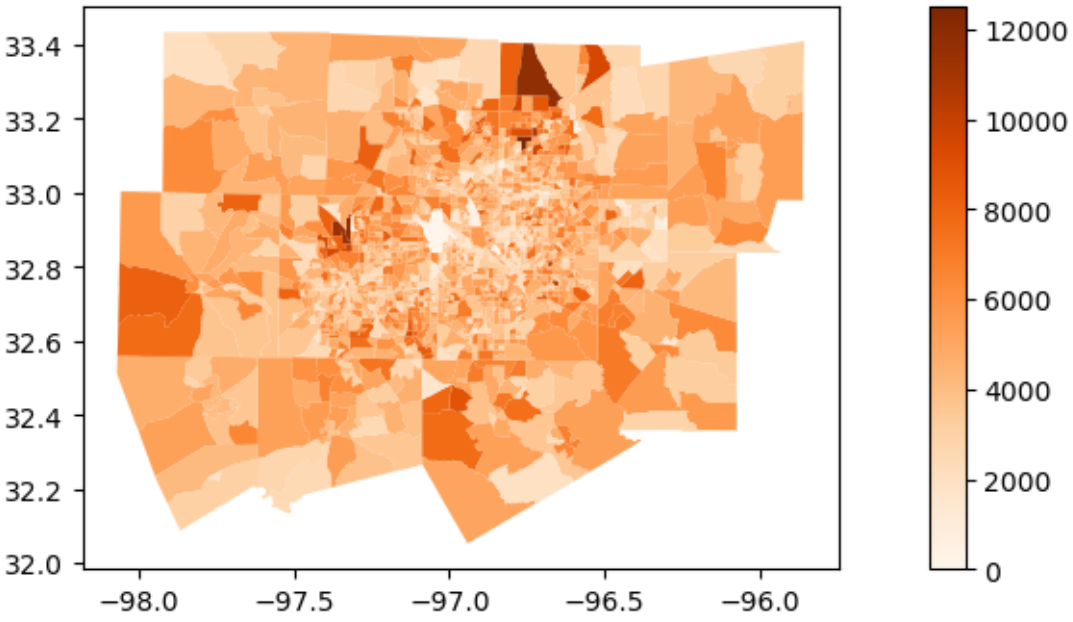
[22]: <Axes: >



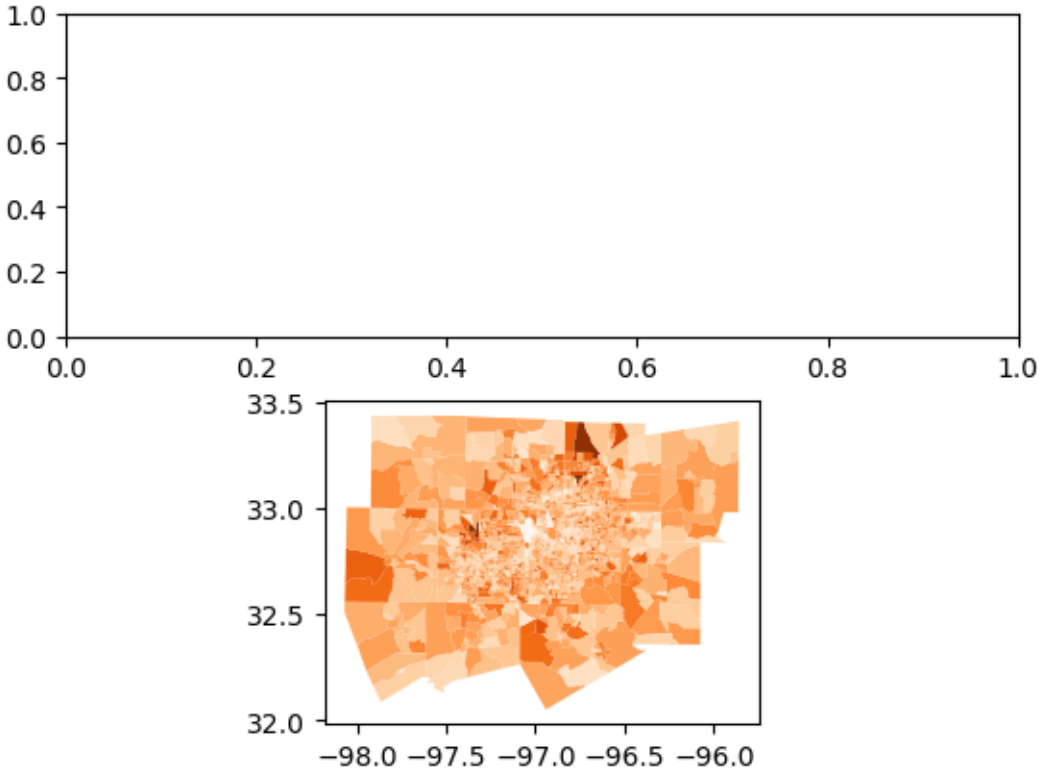
We can turn off the axis for the map by with the `Axes`'s method `set_axis_off()`

[23]:
```
fig, ax = plt.subplots()
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="4%", pad=0.1)
gdf_tract.plot("Pop",legend=True, ax=ax, cax=cax, cmap="Oranges")
```

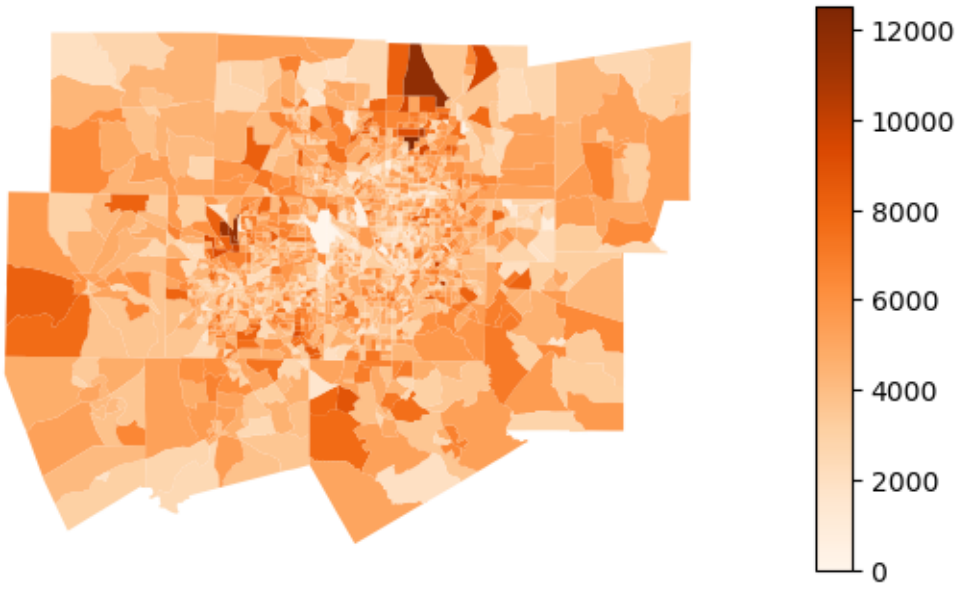
[23]: <Axes: >



[24]:
```
fig, axes = plt.subplots(2, 1)
ax1 = axes[1]
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="4%", pad=0.1)
gdf_tract.plot("Pop",legend=True, ax=ax1, cax=cax, cmap="Oranges")
ax.set_axis_off();
```



[25]:
```
fig, ax = plt.subplots()
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="4%", pad=0.1)
gdf_tract.plot("Pop",legend=True, ax=ax, cax=cax, cmap="Oranges")
ax.set_axis_off();
```

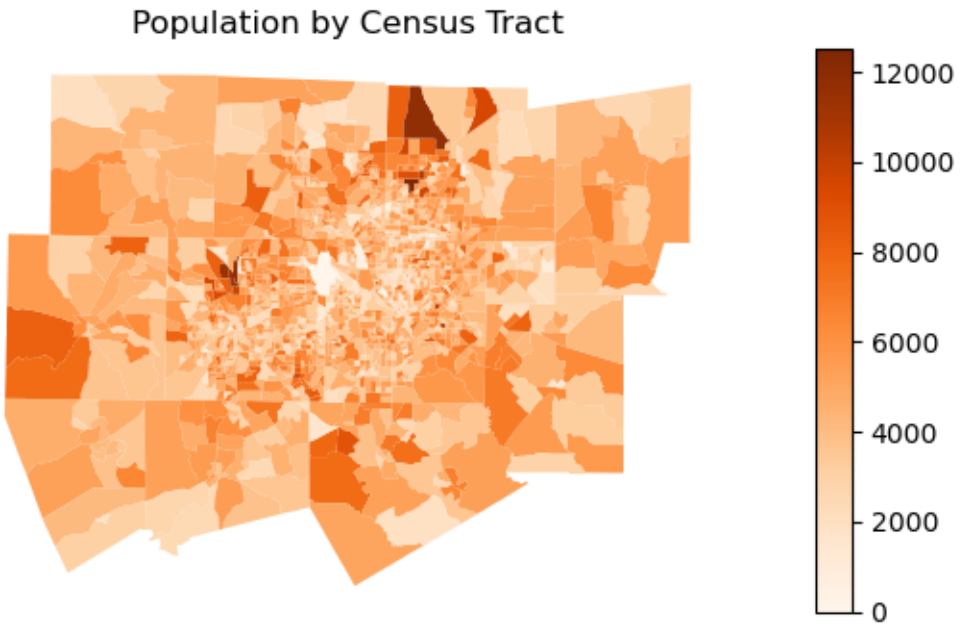


[26]:
```
gdf_tract.columns
```

[26]:
```
Index(['GEOID', 'STATEFP', 'COUNTYFP', 'TRACTCE', 'NAME', 'NAMELSAD', 'MTFCC',
       'FUNCSTAT', 'ALAND', 'AWATER', 'INTPTLAT', 'INTPTLON', 'NumHosp',
       'geometry', 'county', 'Pop', 'White', 'Black', 'Asian', 'Hispanic'],
      dtype='object')
```

[27]:
```
fig, ax = plt.subplots()
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="4%", pad=0.1)
gdf_tract.plot("Pop",legend=True, ax=ax, cax=cax, cmap="Oranges")
ax.set_axis_off();
ax.set_title("Population by Census Tract")
```

[27]:
```
Text(0.5, 1.0, 'Population by Census Tract')
```



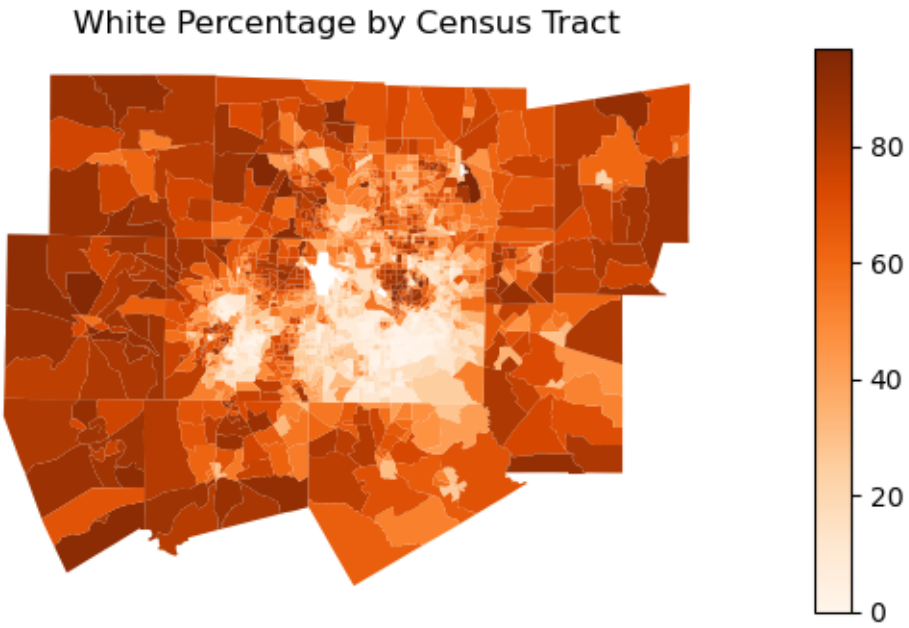
2.1.1 Group exercise

Create a choropleth map (continuous values) for the White percentage

When you are done, raise your hand!

[28]:
```
fig, ax = plt.subplots()
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="4%", pad=0.1)
gdf_tract.plot("White",legend=True, ax=ax, cax=cax, cmap="Oranges")
ax.set_axis_off();
ax.set_title("White Percentage by Census Tract")
```

[28]:
```
Text(0.5, 1.0, 'White Percentage by Census Tract')
```



2.1.2 Aggregate by value and geometry

Currently we have data on census tracts, but we're actually interested in studying patterns at the level of counties.

Obtain a county-level GeoDataFrame:

- county population: `groupby()`
- county geometries: `dissolve()`
  - dissolves all the geometries within a given group together into a single geometric feature
  - aggregates all the rows of data in a group using groupby.aggregate, and

it combines those two results.

[29]:
```
gdf_tract.head()
```

[29]:

	GEOID	STATEFP	COUNTYFP	TRACTCE	NAME	NAMELSAD	MTFCC	\
0	48113015900	48	113	015900	159	Census Tract 159	G5020	
1	48113012604	48	113	012604	126.04	Census Tract 126.04	G5020	
2	48113013010	48	113	013010	130.10	Census Tract 130.10	G5020	
3	48113013622	48	113	013622	136.22	Census Tract 136.22	G5020	
4	48113013621	48	113	013621	136.21	Census Tract 136.21	G5020	

	FUNCSTAT	ALAND	AWATER	INTPTLAT	INTPTLON	NumHosp	\
0	S	5703840	4003497	+32.7362108	-096.9692130	0.0	
1	S	2510513	43527	+32.8428141	-096.6460224	0.0	
2	S	1404466	0	+32.8688147	-096.6802329	0.0	
3	S	1249629	0	+32.9496147	-096.8110809	0.0	

```
4        S   910824      686  +32.9605446  -096.8087578      0.0

                                                geometry  county   Pop  White  \
0  POLYGON ((-96.99498 32.73739, -96.99492 32.737…     113  3611  18.55
1  POLYGON ((-96.65925 32.84716, -96.65785 32.848…     113  7387  21.10
2  POLYGON ((-96.69384 32.87418, -96.69347 32.874…     113  5149  17.58
3  POLYGON ((-96.81880 32.95231, -96.81878 32.952…     113  2811  51.19
4  POLYGON ((-96.81198 32.95712, -96.81197 32.957…     113  4462  35.30

   Black  Asian  Hispanic
0   7.59   1.99     68.73
1  29.48   1.81     46.87
2  32.88   2.23     47.15
3  16.47   2.77     25.15
4  28.91   3.72     26.80
```

[30]:
```
gdf_county = gdf_tract.dissolve(by="COUNTYFP",
                                aggfunc={"NumHosp":'sum',
                                         "Pop":'sum'})
```

[31]:
```
gdf_county.head()
```

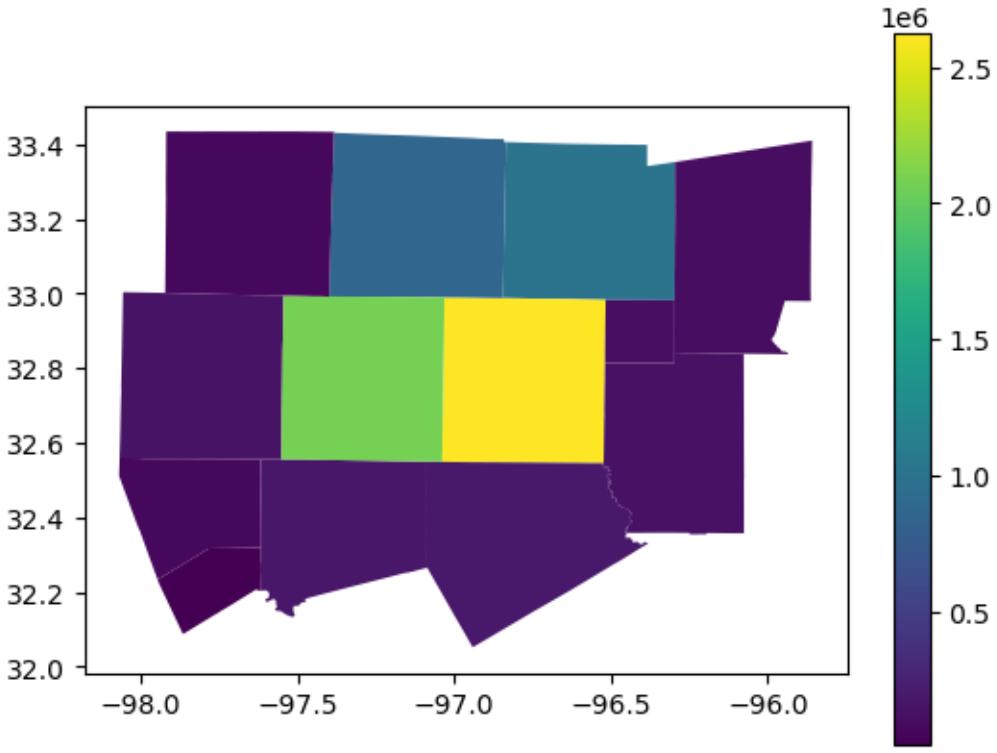
[31]:
```
                                                geometry  NumHosp      Pop
COUNTYFP
085       POLYGON ((-96.78400 32.98666, -96.78441 32.986…      32.0  1006038
113       POLYGON ((-97.03818 32.57979, -97.03813 32.582…      55.0  2622634
121       POLYGON ((-97.12414 32.99030, -97.13024 32.990…      19.0   861690
139       POLYGON ((-97.08705 32.29139, -97.08705 32.292…       3.0   179484
221       POLYGON ((-97.74836 32.31688, -97.74941 32.316…       1.0    60025
```

More summary statatics are allowed for the aggregation function for numerical variables `aggfunc`.

'first', 'last','min','max','sum','mean','median',function,string function name,list of functions and/or function names, e.g. [np.sum, 'mean']

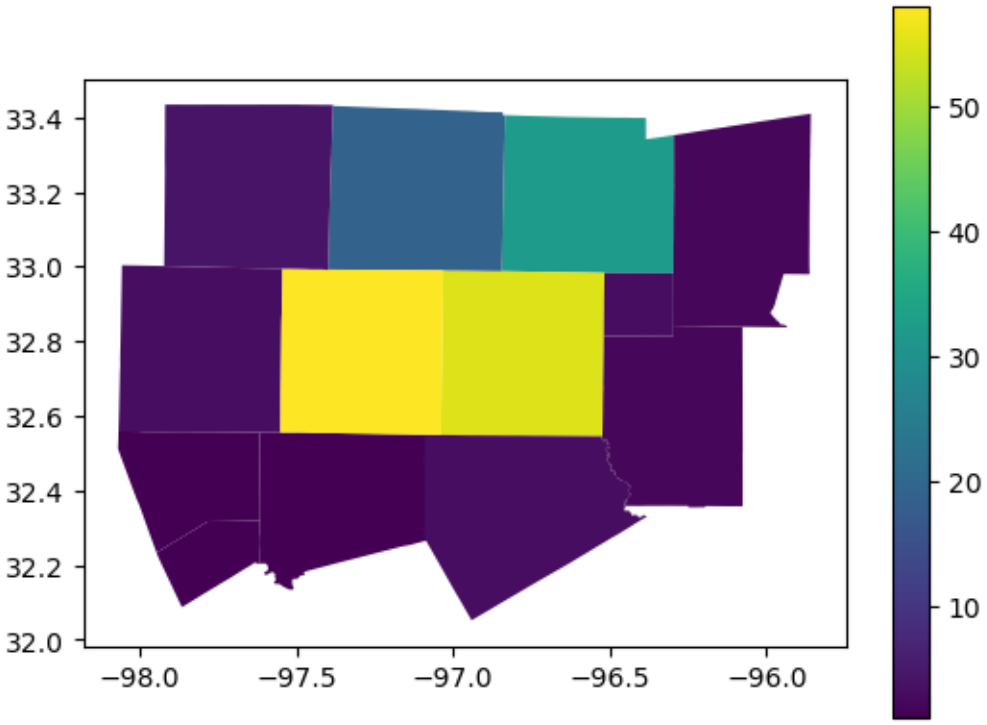
[32]:
```
gdf_county.plot("Pop", legend=True)
```

[32]:
```
<Axes: >
```



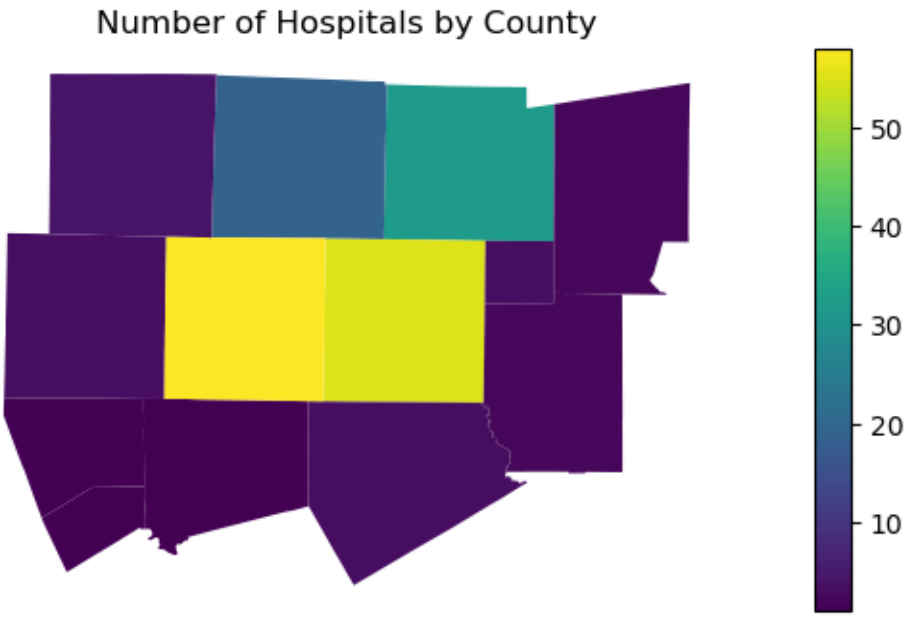
[33]:
```
gdf_county.plot("NumHosp", legend=True)
```

[33]: <Axes: >



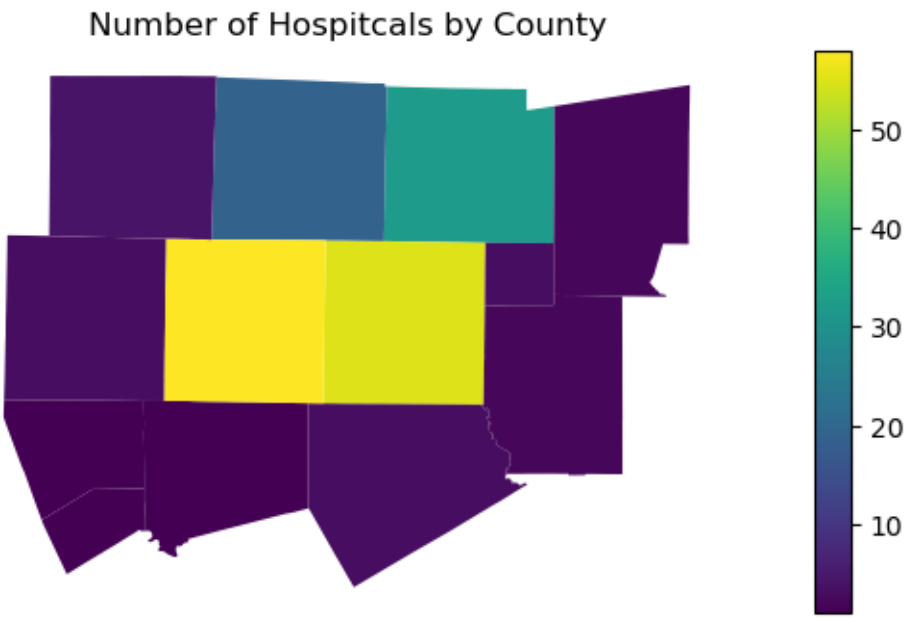
[34]:
```
fig, ax = plt.subplots()
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="4%", pad=0.1)
gdf_county.plot("NumHosp",legend=True, ax=ax, cax=cax)
ax.set_axis_off();
ax.set_title("Number of Hospitals by County")
```

[34]:
```
Text(0.5, 1.0, 'Number of Hospitals by County')
```



Saving a map is similar to saving a figure with functions from matplotlib

[35]:
```
fig, ax = plt.subplots()
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="4%", pad=0.1)
gdf_county.plot("NumHosp",legend=True, ax=ax, cax=cax)
ax.set_axis_off();
ax.set_title("Number of Hospitcals by County")
plt.savefig("NumberHospitalsCountyDallas.png")
```



2.2 Interactive geovisualization

`GeoDataFrame.explore()` method

built on folium/leaflet.js for interactive mapping

[36]:
```
gdf_tract.explore("Pop", legend=True)
```

[36]:
```
<folium.folium.Map at 0x29c18f150>
```

[37]:
```
gdf_tract.explore("Pop", legend=True,
                  tooltip="Pop", # show "Pop" value in tooltip (on hover)
                  popup=True # show all values in popup (on click)
                 )
```

[37]:
```
<folium.folium.Map at 0x2a64b6050>
```

[38]:
```
gdf_tract.explore("Pop", legend=True,
                  tooltip="Pop", # show "Pop" value in tooltip (on hover)
                  popup=True,# show all values in popup (on click)
                  cmap="Oranges" # use "Oranges" matplotlib colormap
                 )
```

[38]:
```
<folium.folium.Map at 0x2a64e1f50>
```

[39]:
```
gdf_tract.explore("Pop", legend=True,
                  tooltip="Pop", # show "Pop" value in tooltip (on hover)
```

```
                    popup=True,# show all values in popup (on click)
                    cmap="Oranges", # use "Oranges" matplotlib colormap
                    style_kwds=dict(color="black") # use black outline
                   )
```

[39]: <folium.folium.Map at 0x2bff884d0>

2.2.1 Background map

we can also change the background map: the default is 'OpenStreetMap Mapnik'.

The current list of built-in providers (when xyzservices is not available):

["OpenStreetMap", "Stamen Terrain", "Stamen Toner", "Stamen Watercolor" "CartoDB positron", "CartoDB dark_matter"]

[40]:
```
gdf_tract.explore("Pop", legend=True,
                    tooltip="Pop", # show "Pop" value in tooltip (on hover)
                    popup=True,# show all values in popup (on click)
                    cmap="Oranges", # use "Oranges" matplotlib colormap
                    style_kwds=dict(color="black"), # use black outline
                    tiles="CartoDB positron", # use "CartoDB positron" tiles
                   )
```

[40]: <folium.folium.Map at 0x29bef7350>

[41]:
```
gdf_tract.explore("Hispanic", legend=True)
```

[41]: <folium.folium.Map at 0x2c72700d0>

3 Further readings

- GeoPandas tutorial on making maps

[ ]: