geopandas_buffering

April 24, 2018

1 Geoprocessing with GeoPandas

Serge Rey

We continue exploring geopandas and more of its geoprocessing capabilities. In this notebook we assume the role of a social scientist who is interested in the topic of environmental equity. They are broadly concerned with the question of whether different racial groups are exposed to different levels of environmental hazards in urban settings.

Their empirical analysis will focus on the case of Riverside County, CA, where the spatial unit of analysis is the Census tract which we encountered and processed in the previous notebook. The researcher will examine the spatial relationships between the highway network and the census tracts to develop operational measures that feed into their environmental equity analysis.

In this notebook we focus on generating new features that will be used in subsequent econometric modeling to test various hypothesis about environmental justice. We want to create new variables that express the exposure to the highway network for census tracts in Riverside, CA.

1.1 Objectives

- Processing polyline shapefiles to represent road networks
- Learn about geographical clipping
- Integrate spatial data sources with different coordinate reference systems
- Apply buffering to derive new features for subsequent analysis

1.2 Setup and Imports

Again we begin with our usual imports:

```
In [1]: %matplotlib inline
        import matplotlib
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: import geopandas as gpd
```

2 Read a LineString Shapefile

Thus far we have encountered two different types of geometries in our shapefiles, namely point and polygons. For our current research, are going to examine the data set "Sanctioned routes for commercial truck traffic located on the state highway system" from the California Department of Transportation. That has been downloaded and stored in the `data` directory.

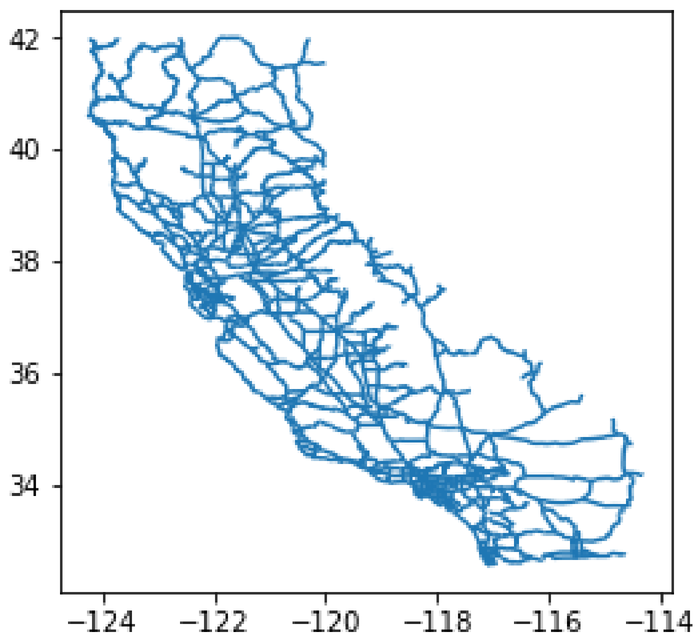
We begin by reading this into a geopandas DataFrame:

```
In [3]: routes_df = gpd.read_file('data/Truck_Route_Network.shp')
```

and taking a view of the features

```
In [4]: routes_df.plot()
```

```
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36ffd2cf8>
```



Futhter exploration reveals the geometries are LineStrings

```
In [5]: routes_df.head()
```

```
Out[5]:    SHAPE_Leng  Beg_Latitu   Beg_Longit  End_Latitu   End_Longit  Route  \
        0    0.426140   33.467051 -117.669910   33.750992 -118.105912      1
        1    0.420058   33.750992 -118.105912   33.931485 -118.395991      1
        2    0.012263   33.931485 -118.395991   33.944521 -118.396115      1
        3    0.101036   33.944521 -118.396115   34.002761 -118.470584      1
        4    0.000513   34.002761 -118.470584   34.014809 -118.486011      1

           District County Beg_PMPre  Beg_PM  \
        0        12    ORA         R   0.129
        1         7     LA      None   0.000
        2         7     LA      None  25.924
```

```
3          7     LA      None  26.897
4          7     LA      None  34.526

                         ...                         Segment_Mi Sp_Restr  \
0                        ...                             33.740     None
1                        ...                             25.858     None
2                        ...                              0.946        R
3                        ...                              6.439     None
4                        ...                              0.050     None

   Rstr_Type  Segmt_Type  KPRA  \
0          0          TA  None
1          0          TA  None
2          5          TA  None
3          0          TA  None
4          0          CL    40

                                          Beg_Locati  \
0                                              Jct 5
1                               Pacific Coast Highway
2                              Jct 105 (Imperial Hwy)
3                                    W. Century Blvd.
4  End Route Break: Lincoln Blvd. near Olympic Av...

                                          End_Locati  \
0                    Orange / Los Angeles County Line
1                              Jct 105 (Imperial Hwy)
2                                    W. Century Blvd.
3  Begin Route Break:  Lincoln Blvd. near Ozone Ave.
4  Lincoln Blvd. at I-10 overcrossing in Santa Mo...

                                             Comment seg_length  \
0                                               None      47438
1                                               None      46761
2  Sign on SB 1 at Century Blvd. says "NO Tank Ve...       1365
3  Rte 1 north of Ozone Ave relinquished to City ...      11247
4                                               None         57

                                            geometry
0  (LINESTRING (-117.670023 33.46687800000004, -1...
1  LINESTRING (-118.10598 33.75104000000003, -118...
2  LINESTRING (-118.396111 33.93223000000003, -11...
3  LINESTRING (-118.396196 33.94486800000003, -11...
4  LINESTRING (-118.485468 34.01443000000002, -11...

[5 rows x 23 columns]
```

```
In [6]: routes_df['geometry'].head()
```

```
Out[6]: 0    (LINESTRING (-117.670023 33.46687800000004, -1...
        1    LINESTRING (-118.10598 33.75104000000003, -118...
        2    LINESTRING (-118.396111 33.93223000000003, -11...
        3    LINESTRING (-118.396196 33.94486800000003, -11...
        4    LINESTRING (-118.485468 34.01443000000002, -11...
        Name: geometry, dtype: object
```

Since we will be using this layer with other spatial datasets, it is good practice to familiarlize ourselves with the Coordinate Reference System:

```
In [7]: routes_df.crs
```

```
Out[7]: {'init': 'epsg:4269'}
```

So the coordinates in our LineStrings are in longitude and latitude.

3 Route Clipping

The researcher has the truck route network for the entire state of California. However, her interest is on the specific case of Riverside County so she needs a way to extract the portions of the network that are within the county. This can be done using the geoprocessing operation *clipping*.

To do this we need to create a layer that will serve to "clip" the road network layer to remove everything outside of Riverside County. We can use the polygon shapefile we created from the previous notebook:

3.1 Read a Polygon Shapefile

```
In [8]: tracts_df = gpd.read_file('data/clinics.shp')
```

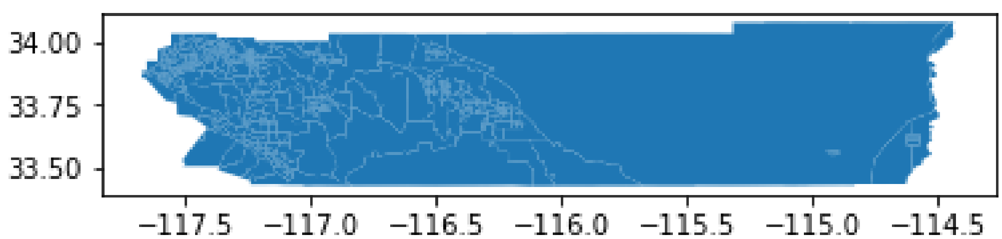
```
In [9]: tracts_df.head()
```

```
Out[9]:        GEOID10           NAMELSAD10     ALAND10  AWATER10   INTPTLAT10  \
        0  06065042012  Census Tract 420.12   2687173.0       0.0  +33.9108776
        1  06065041911  Census Tract 419.11  70257842.0       0.0  +33.7428832
        2  06065041910  Census Tract 419.10  11167489.0   64225.0  +33.7892199
        3  06065040816  Census Tract 408.16   1788821.0       0.0  +33.9024569
        4  06065040815  Census Tract 408.15   1266779.0       0.0  +33.8930776

             INTPTLON10  DP0010001  DP0010002  DP0010003  DP0010004   \
        0  -117.3205065       6242        420        545        620
        1  -117.4957943      10258        840        844        806
        2  -117.4949771       6342        404        453        447
        3  -117.5246107       2594        162        161        227
        4  -117.5114997       3586        231        235        257

                               ...                         DP0210002  DP0210003  \
        0                      ...                              1142        826
        1                      ...                              2881        430
```

```
2                       ...                              1823       350
3                       ...                               688       171
4                       ...                               756       399

   DP0220001  DP0220002  DP0230001  DP0230002  Shape_Leng  Shape_Area  \
0       3927       2299       3.44       2.78    0.095958    0.000262
1       8710       1543       3.02       3.59    0.466106    0.006836
2       5177       1165       2.84       3.33    0.200974    0.001093
3       2133        451       3.10       2.64    0.082444    0.000174
4       2462       1124       3.26       2.82    0.050637    0.000123

   clinics                                           geometry
0      0.0  POLYGON ((-117.300465 33.91310800000002, -117...
1      0.0  POLYGON ((-117.5101979999999 33.800273, -117.5...
2      0.0  POLYGON ((-117.5029849999999 33.82494899999995...
3      0.0  POLYGON ((-117.515118 33.90096800000009, -117...
4      0.0  POLYGON ((-117.503863 33.89735700000011, -117...

[5 rows x 196 columns]
```

```
In [10]: tracts_df.plot()
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36fd32048>
```



3.2 Get routes intersecting Riverside County

To select only the routes within Riverside County we could take several approaches. We have the tract layer for the county that has 453 tracts, as well as the road network layer for the state. That has 966 segments. We could then use the intersects method for each tract to test if it intersects with a particular segment of the road network, and then keep all the segments where we find an intersection with the tract.

While this would work, it turns out to be very inefficient as a brute force approach would require we compare each of 453 tracts against each of 966 segments and test for an intersection.

We can do better.

If we think about our problem from a slightly different perspective, we know that if we find a segment that intersects with a tract within Riverside county, it must, by definition, intersect with the County polygon, if we had such a thing.

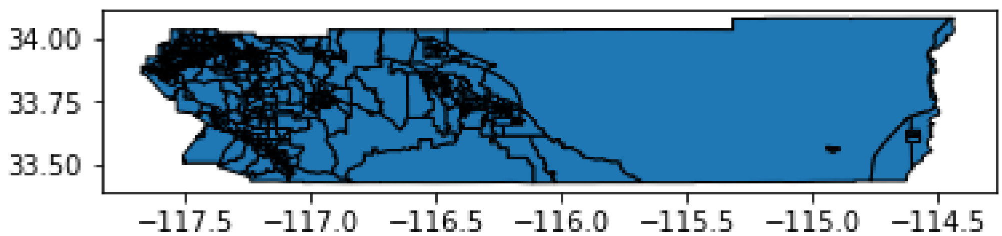
This would substantially reduce the number of intersection tests (or more broadly, "hit tests") we need to conduct. Rather than having to compare 453 tract polygons with 966 road segments, we now only need compare 1 polygon against each of the road segments. That is a 453X reduction in computation. Nice.

3.2.1 Dissolve

Ok, but we do not yet have the magical county polygon. It seems worth it to get one, and using another method of the geopandas DataFrame for the tracts, we can. First, we can re-examine our DataFrame:

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

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36fcd06d8>
```



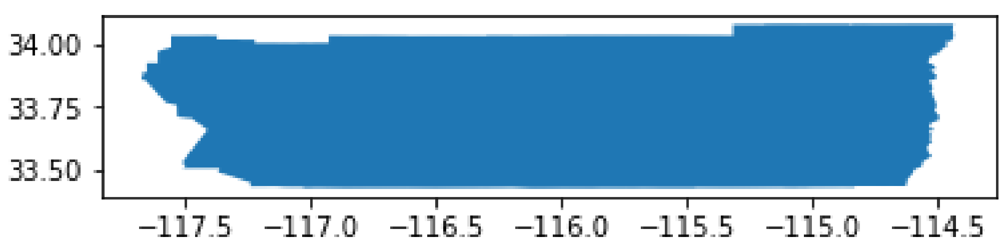
What we are going to do is dissolve all the tract boundaries that do not coincide with the boundary of the DataFrame's geometry collection.

This is done by creating a new attribute that takes on the same values for each feature, and calling the `dissovle` method with that attribute as the argument to the `by` option:

```
In [12]: tracts_df['dummy'] = 1.0
         county = tracts_df.dissolve(by='dummy')
```

```
In [13]: county.plot()
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36fc1fe48>
```



```
In [14]: county.shape

Out[14]: (1, 196)
```

Note that we could have also obtained this polygon by using the `unary_union` method of the GeoSeries:

```
In [15]: county_uu = tracts_df['geometry'].unary_union
         county_uu

Out[15]:
```

This gives us a Shapely Polygon. We would then have toconstruct a new GeoDataFrame with this as the Geometry column. Instead, we will continue with the `county` DataFrame obtained from the dissolve operation since this saves us one step. (We simply note the $\text{unary}_{\text{union}}$ as you never know when you may need it.)

We now have our single polygon for the county.

In our earlier notebook we saw that care needs to be taken when testing for intersections between features from two different DataFrames, as this is done on an element-wise basis.

There are a couple of ways to handle this. First, using what are known as **lambdas**:

```
In [16]: r = routes_df['geometry']

In [17]: type(r)

Out[17]: geopandas.geoseries.GeoSeries

In [18]: r.apply(lambda x: x.intersects(county.iloc[0]['geometry']))

Out[18]: 0      False
         1      False
         2      False
         3      False
         4      False
         5      False
         6      False
         7      False
         8      False
         9      False
         10     False
         11     False
```

```
12      False
13      False
14      False
15      False
16      False
17      False
18      False
19      False
20      False
21      False
22      False
23      False
24      False
25      False
26      False
27      False
28      False
29      False
        ...
936     False
937     False
938     False
939     False
940     False
941     False
942     False
943     False
944     False
945     False
946     False
947     False
948     False
949     False
950     False
951     False
952     False
953     False
954     False
955     False
956     False
957     False
958     False
959     False
960     False
961     False
962     False
963     False
964     False
```

```
965    False
Name: geometry, Length: 966, dtype: bool
```

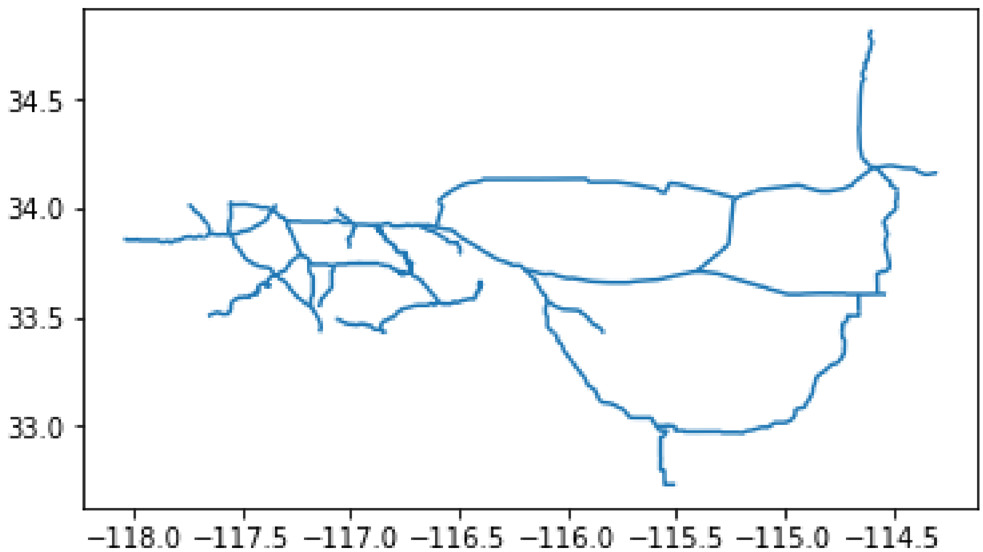
```
In [19]: rc_routes = r[r.apply(lambda x: x.intersects(county.iloc[0]['geometry']))]
```

```
In [20]: rc_routes.shape
```

```
Out[20]: (42,)
```

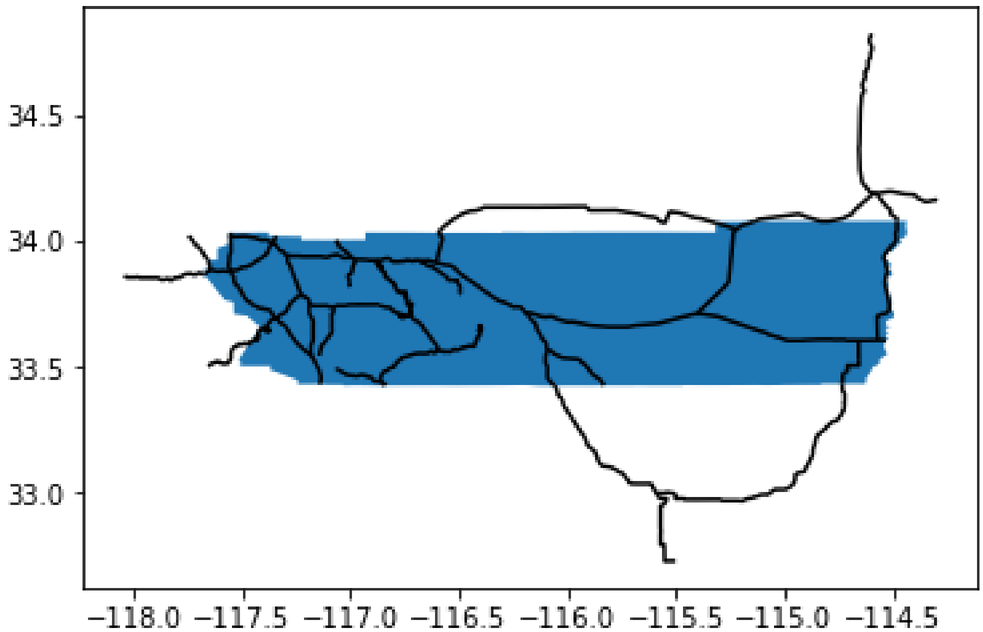
```
In [21]: rc_routes.plot()
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36fc07f60>
```



Plotting the two layers to see what we are now working with gives us:

```
In [22]: ax = plt.gca()
         rc_routes.plot(ax=ax, edgecolor='k')
         county.plot(ax=ax)
         plt.show()
```



Lambdas are handy, but tend to make code a little more difficult to read. Technically they are known as "anonymous functions". A more transparent approach is to use a simple loop and test each route segment for intersection with the county, and append the segment to a list to store all the segments that intersect with the county:

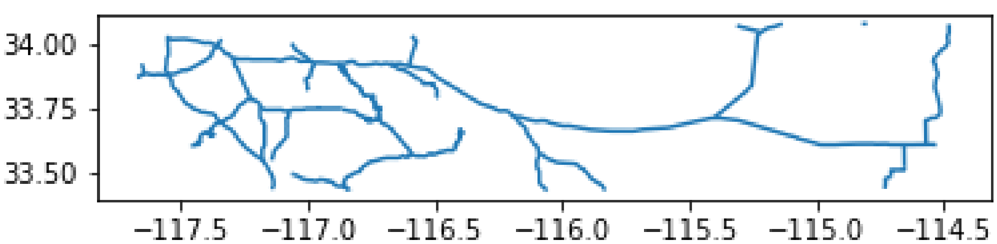
```
In [23]: geoms = []
         for idx, route in enumerate(rc_routes):
             print(idx)
             geoms.append(route.intersection(county.iloc[0]['geometry']))
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

Now we use this Python list of intersection objects (which are segments) into a GeoSeries:

```
In [24]: rc_hw = gpd.GeoSeries(geoms)
         rc_hw.plot()
```

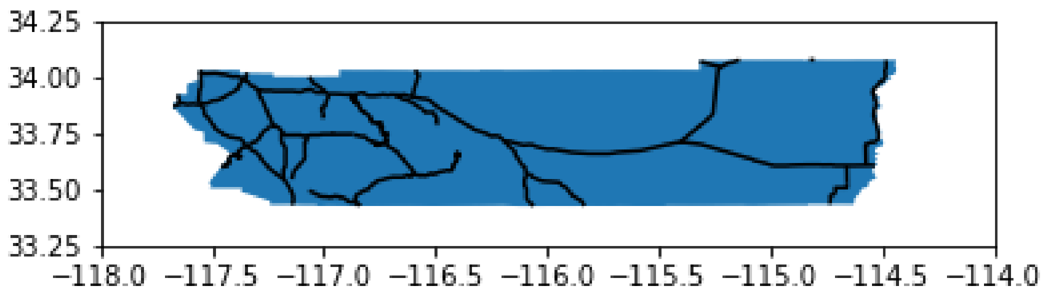
```
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36d582b70>
```



and, plot the new series with our county polygon:

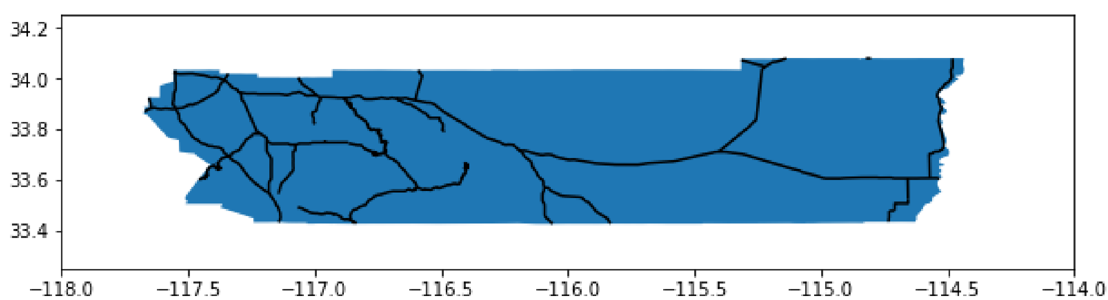
```
In [25]: ax = plt.gca()
         county.plot(ax=ax)
```

```
rc_hw.plot(ax=ax, edgecolor='k')
ax.set_xlim(-118.0, -114.0); ax.set_ylim(33.25, 34.25)
ax.set_aspect('equal')
plt.show()
```

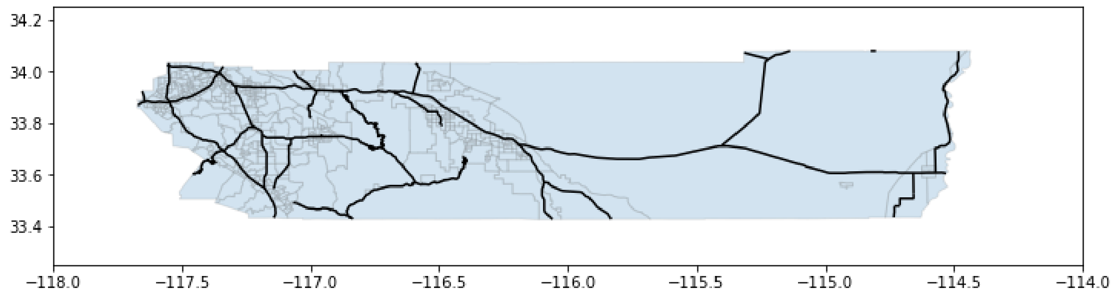


we set the limits for the horizontal and vertical axes to zoom in. We can also change the plot size:

```
In [26]: plt.rcParams['figure.figsize'] = (10, 8)
         ax = plt.gca()
         county.plot(ax=ax)
         rc_hw.plot(ax=ax, edgecolor='k')
         ax.set_xlim(-118.0, -114.0); ax.set_ylim(33.25, 34.25)
         ax.set_aspect('equal')
         plt.show()
```




```
In [27]: plt.rcParams['figure.figsize'] = (12, 10)
         ax = plt.gca()
         tracts_df.plot(ax=ax, edgecolor='grey', alpha=0.2)
         rc_hw.plot(ax=ax, edgecolor='k')
         ax.set_xlim(-118.0, -114.0); ax.set_ylim(33.25, 34.25)
         ax.set_aspect('equal')
         plt.show()
```



And finally, let us create a DataFrame from the GeoSeries:

```
In [28]: type(rc_hw)

Out[28]: geopandas.geoseries.GeoSeries

In [29]: rc_hw = gpd.GeoDataFrame({'geometry': rc_hw})

In [30]: rc_hw.shape

Out[30]: (42, 1)

In [31]: tracts_df.shape

Out[31]: (453, 197)
```

4 Spatial Joins: Which Tracts Intersect the Truck Network?

We now have the truck route network clipped to the extent of Riverside County. Using this layer, we can determine which census tracts intersect the network within the county. For this, we revisit the concept of a spatial join. There are different flavors of spatial joins that can be used in practice. Here we explore the options before deciding which one serves our particular need best.

We begin with a so called "inner" join:

```
In [32]: # spatial join, tracts with roads
         tracts_with_roads = gpd.sjoin(tracts_df, rc_hw, how='inner', op='intersects')

Warning: CRS does not match!
```

We see the warning about the CRS mismatch. Let us see what is going on:

```
In [33]: tracts_df.crs

Out[33]: {'init': 'epsg:4269'}
```

and

```
In [34]: rc_hw.crs
```

So the route DataFrame does not have a CRS. We can correct this by setting it to that of the tracts data frame:

```
In [35]: rc_hw.crs = tracts_df.crs # create a crs for the rc_hw
         rc_hw = rc_hw.to_crs(tracts_df.crs) # update the coordinates accordingly
```

and when we repeat the join:

```
In [36]: # spatial join, tracts with roads
         tracts_with_roads = gpd.sjoin(tracts_df, rc_hw, how='inner', op='intersects')
```

Silence is golden.

Now we can see what our join operation has returned. We stored the results in a new object:

```
In [37]: tracts_with_roads.head()
```

```
Out[37]:        GEOID10            NAMELSAD10      ALAND10   AWATER10    INTPTLAT10  \
         1  06065041911  Census Tract 419.11  70257842.0       0.0  +33.7428832
         2  06065041910  Census Tract 419.10  11167489.0   64225.0  +33.7892199
         6  06065040813  Census Tract 408.13  12539455.0    3687.0  +33.9155438
         7  06065040812  Census Tract 408.12   3427721.0       0.0  +33.9244565
         8  06065040616  Census Tract 406.16   8459218.0  213354.0  +33.9422906

              INTPTLON10  DP0010001  DP0010002  DP0010003  DP0010004     ...        \
         1  -117.4957943      10258        840        844        806     ...
         2  -117.4949771       6342        404        453        447     ...
         6  -117.5321545       6080        380        417        498     ...
         7  -117.5494884       3480        217        236        237     ...
         8  -117.5776719       7610        697        702        710     ...

            DP0220001  DP0220002  DP0230001  DP0230002  Shape_Leng  Shape_Area  \
         1       8710       1543       3.02       3.59    0.466106    0.006836
         2       5177       1165       2.84       3.33    0.200974    0.001093
         6       5345        730       3.40       3.56    0.218446    0.001223
         7       2615        857       3.30       2.83    0.082574    0.000334
         8       6436       1174       4.06       4.91    0.197679    0.000846

            clinics                                           geometry  dummy  \
         1      0.0  POLYGON ((-117.5101979999999 33.800273, -117.5...    1.0
         2      0.0  POLYGON ((-117.5029849999999 33.82494899999995...    1.0
         6      0.0  POLYGON ((-117.5140629999999 33.90922599999999...    1.0
         7      0.0  POLYGON ((-117.5367549999999 33.9281160000001,...    1.0
         8      0.0  POLYGON ((-117.566953 33.9608540000001, -117.5...    1.0

            index_right
         1            1
         2            1
```

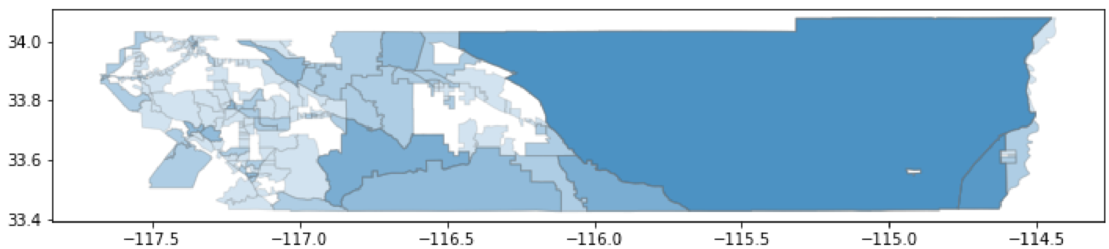
```
6              1
7              1
8              1

[5 rows x 198 columns]
```

If we scroll to the right of the DataFrame output, we see a column labeled $\textbf{index}_{\textbf{right}}$. The values in this column indicate the index of the features in the right DataFrame (in our case the road network) that intersect with the feature in the current row of the left DataFrame (the tracts).

Plotting the resulting DataFrame we see:

```
In [38]: tracts_with_roads.plot(edgecolor='grey', alpha=0.2)
```

```
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36c489978>
```



Close inspection reveals some missing tracts. What is going on here?

```
In [39]: tracts_with_roads.shape
```

```
Out[39]: (256, 198)
```

We see there are 256 features in our new DataFrame resulting from the join. But this is less than the number of tracts in the county:

```
In [40]: tracts_df.shape
```

```
Out[40]: (453, 197)
```

So our plot is not incorrect. It is giving us what we asked for - a plot of the DataFrame for the tracts that intersect the truck network.

A second type of join can be obtained by setting the `how` option to `'left'`:

```
In [41]: # spatial join, tracts with roads
         tracts_with_roads = gpd.sjoin(tracts_df, rc_hw, how='left', op='intersects')
```

This overwrites the resulting DataFrame, so the number of features changes:

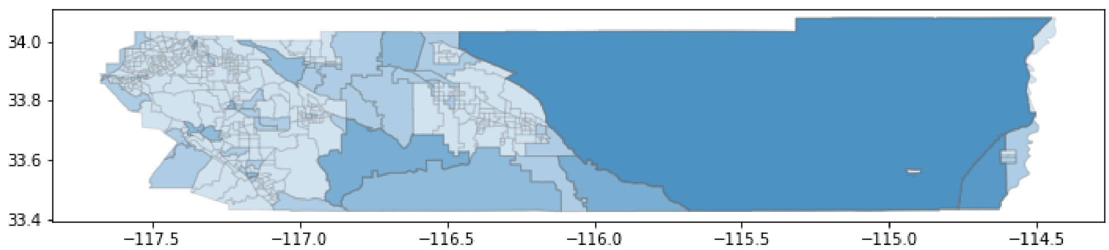
```
In [42]: tracts_with_roads.shape
```

```
Out[42]: (526, 198)
```

This is a larger number than the number of tracts. What is going on?

```
In [43]: ## 'how=left' uses keys from left_df and retains left_df geometry
         # shows all tracts with or withing intersection with network
         tracts_with_roads.plot(edgecolor='grey', alpha=0.2)
```

```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36d5ac0f0>
```



The plot doesn't suggest anything fishy. More introspection:

```
In [44]: tracts_with_roads.head()
```

```
Out[44]:        GEOID10          NAMELSAD10     ALAND10  AWATER10   INTPTLAT10  \
         0  06065042012  Census Tract 420.12   2687173.0       0.0  +33.9108776
         1  06065041911  Census Tract 419.11  70257842.0       0.0  +33.7428832
         2  06065041910  Census Tract 419.10  11167489.0   64225.0  +33.7892199
         3  06065040816  Census Tract 408.16   1788821.0       0.0  +33.9024569
         4  06065040815  Census Tract 408.15   1266779.0       0.0  +33.8930776

              INTPTLON10  DP0010001  DP0010002  DP0010003  DP0010004     ...        \
         0  -117.3205065       6242        420        545        620     ...
         1  -117.4957943      10258        840        844        806     ...
         2  -117.4949771       6342        404        453        447     ...
         3  -117.5246107       2594        162        161        227     ...
         4  -117.5114997       3586        231        235        257     ...

            DP0220001  DP0220002  DP0230001  DP0230002  Shape_Leng  Shape_Area  \
         0       3927       2299       3.44       2.78    0.095958    0.000262
         1       8710       1543       3.02       3.59    0.466106    0.006836
         2       5177       1165       2.84       3.33    0.200974    0.001093
         3       2133        451       3.10       2.64    0.082444    0.000174
         4       2462       1124       3.26       2.82    0.050637    0.000123

            clinics                                         geometry  dummy  \
         0      0.0  POLYGON ((-117.300465 33.91310800000002, -117...    1.0
         1      0.0  POLYGON ((-117.5101979999999 33.800273, -117.5...    1.0
         2      0.0  POLYGON ((-117.5029849999999 33.82494899999995...    1.0
         3      0.0  POLYGON ((-117.515118 33.90096800000009, -117...    1.0
```

```
4      0.0  POLYGON ((-117.503863 33.89735700000011, -117...    1.0

   index_right
0          NaN
1          1.0
2          1.0
3          NaN
4         29.0

[5 rows x 198 columns]
```

Again, scrolling to the right we see the $\textbf{index}_{\textbf{right}}$ column, but now we see a mixture of `NaN` and numerical values. The `NaN` values appear in rows for tracts that do not intersect the road network. Hence there is no feature in the right DataFrame that intersects with that feature in the left DataFrame.

But, this doesn't explain why we have more features in the resulting DataFrame than in the left data frame. Something else must be happening. And it is:

```
In [45]: len(tracts_with_roads['GEOID10'].unique())

Out[45]: 453
```

We have the correct number of unique geographic identifiers. Using these we can determine how many records we have for each unique identifier (tract):

```
In [46]: tracts_with_roads.groupby(['GEOID10']).size()

Out[46]: GEOID10
         06065030101    2
         06065030103    1
         06065030104    2
         06065030200    1
         06065030300    1
         06065030400    1
         06065030501    1
         06065030502    3
         06065030503    1
         06065030601    1
         06065030602    1
         06065030603    1
         06065030700    1
         06065030800    1
         06065030900    1
         06065031001    1
         06065031002    1
         06065031100    1
         06065031200    1
         06065031300    1
         06065031401    1
```

```
06065031402    1
06065031501    1
06065031502    1
06065031601    1
06065031602    1
06065031701    1
06065031702    1
06065031703    1
06065031704    1
              ..
06065049400    1
06065049500    1
06065049600    2
06065049700    1
06065049800    1
06065050300    2
06065050400    1
06065050500    3
06065050600    1
06065050700    1
06065050900    2
06065051100    1
06065051200    1
06065051300    1
06065051400    1
06065940100    1
06065940400    2
06065940500    1
06065940600    1
06065940700    1
06065940800    1
06065940900    1
06065941000    2
06065941100    1
06065941200    1
06065941300    1
06065941400    1
06065941500    1
06065980004    1
06065981000    1
Length: 453, dtype: int64
```

Ah, there are some tracts that appear multiple times in the resulting DataFrame. We can examine one of these using

```
In [47]: tracts_with_roads[tracts_with_roads['GEOID10']=='06065030502']

Out[47]:           GEOID10           NAMELSAD10    ALAND10  AWATER10   INTPTLAT10  \
         140  06065030502  Census Tract 305.02  1914213.0       0.0  +33.9857419
```

```
140  06065030502  Census Tract 305.02  1914213.0       0.0  +33.9857419
140  06065030502  Census Tract 305.02  1914213.0       0.0  +33.9857419

        INTPTLON10  DP0010001  DP0010002  DP0010003  DP0010004     ...       \
140  -117.3560514       2220        223        183        188     ...
140  -117.3560514       2220        223        183        188     ...
140  -117.3560514       2220        223        183        188     ...

     DP0220001  DP0220002  DP0230001  DP0230002  Shape_Leng  Shape_Area  \
140        969       1127       4.85        4.1    0.078829    0.000187
140        969       1127       4.85        4.1    0.078829    0.000187
140        969       1127       4.85        4.1    0.078829    0.000187

     clinics                                           geometry  dummy  \
140      0.0  POLYGON ((-117.359873 33.99014200000011, -117...    1.0
140      0.0  POLYGON ((-117.359873 33.99014200000011, -117...    1.0
140      0.0  POLYGON ((-117.359873 33.99014200000011, -117...    1.0

     index_right
140         29.0
140         39.0
140          3.0

[3 rows x 198 columns]
```

and scrolling over to the right of the output cell reveals that the tract with the GEOID10 of 06065030502 intersects with three different segments of the road network: 29.0, 39.0, and 3.

What has happen is the 'left' join keeps all of the features from the left database and reports either an `NaN` value, or each unique intersection between the tract and a particular segment of the road network. In other words, there will be at least as many features in the resulting DataFrame as in the left DataFrame. There will be more when one or more features from the left data frame intersects with more than a single feature from the right DataFrame.

Thus far we have examined a "inner" join and a "left" join. The final option is a "right" join:

```
In [48]: # spatial join, tracts with roads
         tracts_with_roads = gpd.sjoin(tracts_df, rc_hw, how='right', op='intersects')
```

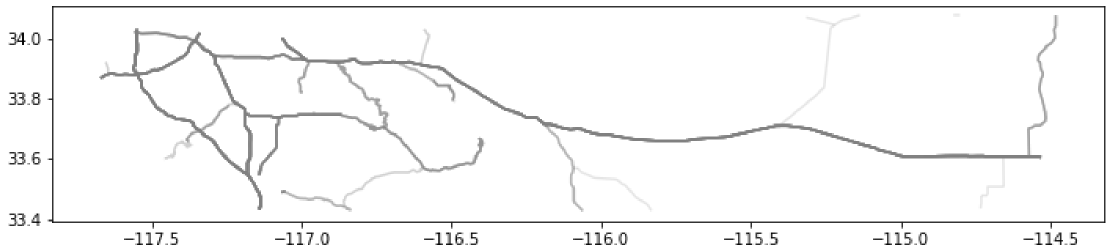
```
In [49]: tracts_with_roads.shape
```

```
Out[49]: (256, 198)
```

There is that number again: 256. What is happening here?

```
In [50]: ## 'how=right' uses keys from right DataFrame and retains right df geometry
         tracts_with_roads.plot(edgecolor='grey', alpha=0.2)
```

```
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36d582518>
```



These are not tracts but rather the LineStirngs. What is happening is that a `right` join keeps each of the features from the right DataFrame and lists each unique intersection with a feature from the left DataFrame:

So, if we are interested in the question of whether tracts intersecting the highway network are different from those not interseting the highways, which one do we want?

There are several ways we could do this, but the approach we take here is to use the inner join:

```
In [51]: tracts_with_roads = gpd.sjoin(tracts_df, rc_hw, how='inner', op='intersects')

In [52]: tracts_with_roads.shape

Out[52]: (256, 198)
```

With this in hand, we can create an indicator variable for use in subsequent analysis. Here the indicator will be 1 if the tract intersects one or more route segments, and zero other wise:

```
In [53]: # Let's create an indicator (dummy) variable for use later
         import numpy as np
         geoids = tracts_df['GEOID10'].values
         tract_hw = np.array([geoid in tracts_with_roads['GEOID10'].values for geoid in geoids])

         tract_hw

Out[53]: array([False,  True,  True, False,  True, False,  True,  True,  True,
                 True, False,  True, False,  True,  True, False,  True, False,
                False, False, False, False, False,  True,  True,  True, False,
                False,  True, False, False,  True, False, False,  True,  True,
                False, False, False,  True, False, False, False, False, False,
                False, False, False, False,  True, False, False, False, False,
                 True,  True,  True, False,  True, False, False,  True,  True,
                 True, False,  True, False, False, False, False, False, False,
                 True,  True, False,  True,  True, False, False, False, False,
                 True,  True,  True,  True,  True,  True, False, False, False,
                False, False, False, False, False, False, False, False,  True,
                False,  True, False, False,  True,  True,  True, False,  True,
                False, False,  True,  True,  True, False, False, False,  True,
                False, False, False, False, False, False, False,  True, False,
                False,  True,  True, False, False, False, False,  True, False,
```

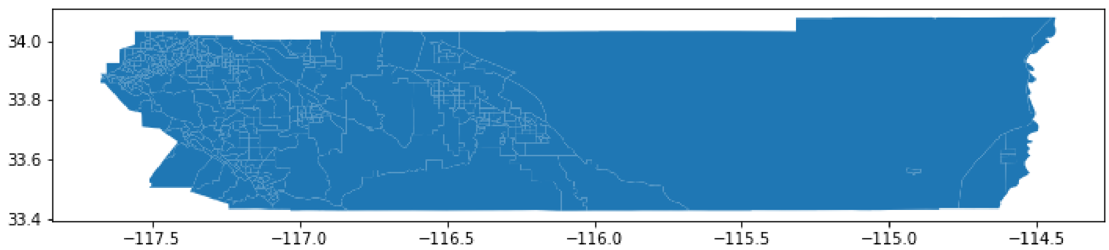
```
False,  True,  True, False, False,  True, False, False, False,
False, False,  True, False, False, False,  True,  True, False,
 True, False, False, False,  True,  True,  True,  True,  True,
 True, False,  True,  True, False, False, False, False, False,
False,  True,  True,  True,  True, False,  True, False,  True,
 True, False,  True, False, False,  True, False, False, False,
False,  True,  True,  True, False, False, False, False, False,
False, False, False, False, False, False, False, False, False,
False,  True, False, False,  True, False,  True, False, False,
False, False,  True, False, False, False, False, False, False,
 True,  True,  True, False, False,  True,  True,  True, False,
False, False, False,  True, False, False, False,  True,  True,
 True, False, False, False, False,  True,  True, False, False,
False, False,  True, False,  True, False,  True,  True, False,
False,  True, False, False, False,  True,  True, False, False,
False, False, False, False,  True, False,  True,  True, False,
 True, False, False,  True,  True, False, False, False, False,
 True, False, False, False, False,  True, False, False, False,
 True,  True,  True, False, False, False, False, False, False,
 True,  True,  True, False,  True,  True,  True,  True, False,
 True,  True, False, False, False, False,  True, False,  True,
 True, False,  True, False,  True,  True,  True,  True, False,
False,  True,  True,  True, False, False, False,  True,  True,
False,  True, False, False, False,  True, False,  True, False,
 True, False,  True, False,  True,  True,  True, False,  True,
 True, False, False, False,  True, False, False,  True, False,
False,  True, False,  True, False, False,  True, False,  True,
 True,  True, False, False, False, False, False,  True,  True,
False,  True,  True,  True, False, False, False, False, False,
False, False, False,  True, False,  True,  True, False, False,
False,  True,  True, False,  True,  True, False, False,  True,
False,  True, False, False,  True,  True,  True,  True,  True,
False,  True, False, False, False, False, False, False,  True,
 True,  True, False, False,  True, False, False,  True,  True,
 True, False,  True, False, False, False, False, False, False,
 True,  True, False])
```

We convert the Boolean valued array into a numerical type and store it in our indicator variable `intersectshw` in our tract DataFrame:

```
In [54]: tracts_df['intersectshw'] = tract_hw*1.
```

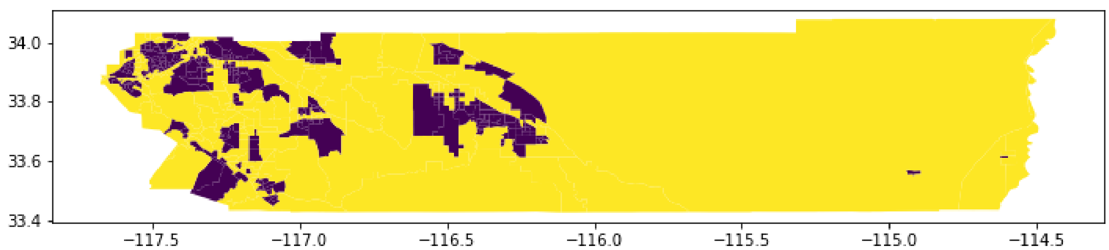
We can now visualize our work:

```
In [55]: tracts_df.plot()

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36d815e80>
```



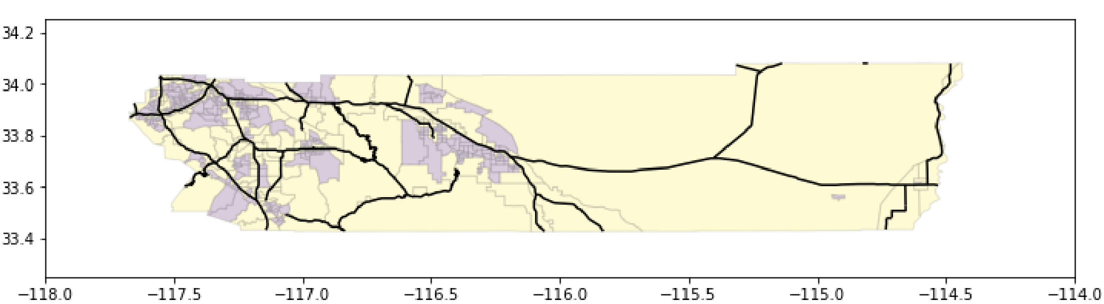
That plots the entire DataFrame. We would like to distinguish tracts that intersect the network from those that do not:

```
In [56]: tracts_df.plot(column='intersectshw')

Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36d84c630>
```



Great, but which color represents the tract intersecting the network? We can tighten up this visualization:

```
In [57]: plt.rcParams['figure.figsize'] = (12, 10)
         ax = plt.gca()
         tracts_df.plot(ax=ax, column='intersectshw',edgecolor='grey', alpha=0.2)
         rc_hw.plot(ax=ax, edgecolor='k')
         ax.set_xlim(-118.0, -114.0); ax.set_ylim(33.25, 34.25)
         ax.set_aspect('equal')
         plt.show()
```



And we see the results of our geoprocessing.

We can save our DataFrame by writing it out to a shapefile for future analysis.

```
In [58]: # save our work to an augmented shapefile
         tracts_df.to_file('data/tracts_routes.shp')
```

5 Spatial Joins: Take Two

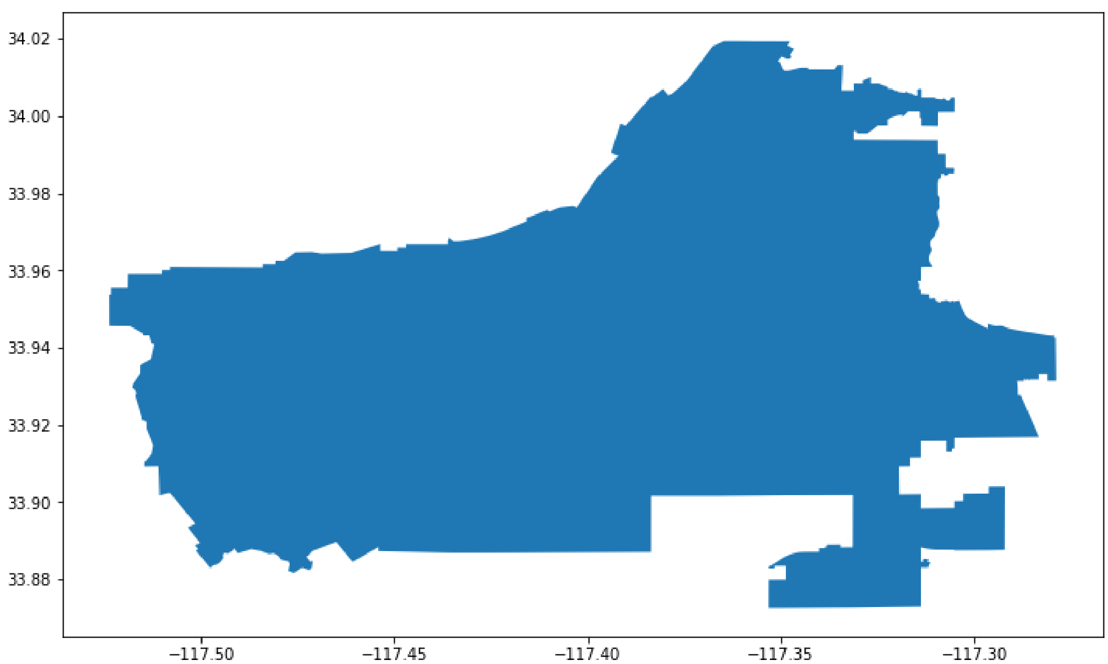
Our social scientist is pretty happy with what she has been able to accomplish with Geopandas and its geoprocessing.

Taking advantage of these new skills, she wants to further refine the scope of her analysis as she realizes much of the eastern part of the county consists of very large census tracts with low population. So she decides to focus only on the case of the City of Riverside.

She has obtained a shapefile for the official city boundaries from the California Department of Transportation:

```
In [59]: city = gpd.read_file('data/riverside_city.shp')

         city.plot()
```

```
Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb3677cfac8>
```



And she uses this to do a spatial join to determine which tracts in Riverside County are within Riverside City:

```
In [60]: city_tracts = gpd.sjoin(tracts_df, city, how='inner', op='intersects')

         city_tracts.head()
```

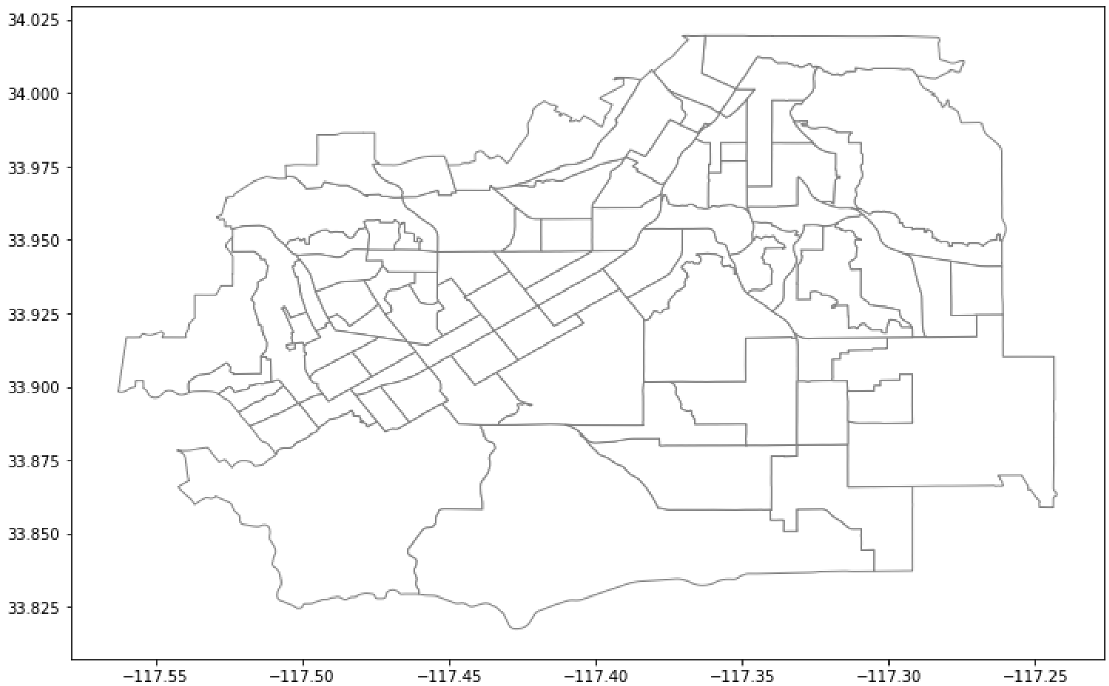
```
Out[60]:        GEOID10           NAMELSAD10     ALAND10  AWATER10   INTPTLAT10  \
         0  06065042012  Census Tract 420.12   2687173.0       0.0  +33.9108776
         3  06065040816  Census Tract 408.16   1788821.0       0.0  +33.9024569
         4  06065040815  Census Tract 408.15   1266779.0       0.0  +33.8930776
         5  06065040814  Census Tract 408.14   1088363.0       0.0  +33.8973552
         6  06065040813  Census Tract 408.13  12539455.0    3687.0  +33.9155438

              INTPTLON10  DP0010001  DP0010002  DP0010003  DP0010004    ...     \
         0  -117.3205065       6242        420        545        620    ...
         3  -117.5246107       2594        162        161        227    ...
         4  -117.5114997       3586        231        235        257    ...
         5  -117.5175804       4782        392        362        392    ...
         6  -117.5321545       6080        380        417        498    ...

            index_right       NAME  CityType  Pop2010  Land_sqmi  DateIncorp  \
         0            0  Riverside      City   303871      81.14  1883-10-11
         3            0  Riverside      City   303871      81.14  1883-10-11
         4            0  Riverside      City   303871      81.14  1883-10-11
         5            0  Riverside      City   303871      81.14  1883-10-11
         6            0  Riverside      City   303871      81.14  1883-10-11

                                 WebLink     County  Notes  CityAbbv
         0  http://www.riversideca.gov  Riverside   None       Riv
         3  http://www.riversideca.gov  Riverside   None       Riv
         4  http://www.riversideca.gov  Riverside   None       Riv
         5  http://www.riversideca.gov  Riverside   None       Riv
         6  http://www.riversideca.gov  Riverside   None       Riv

         [5 rows x 208 columns]
```

```
In [61]: city_tracts.shape
```

```
Out[61]: (84, 208)
```

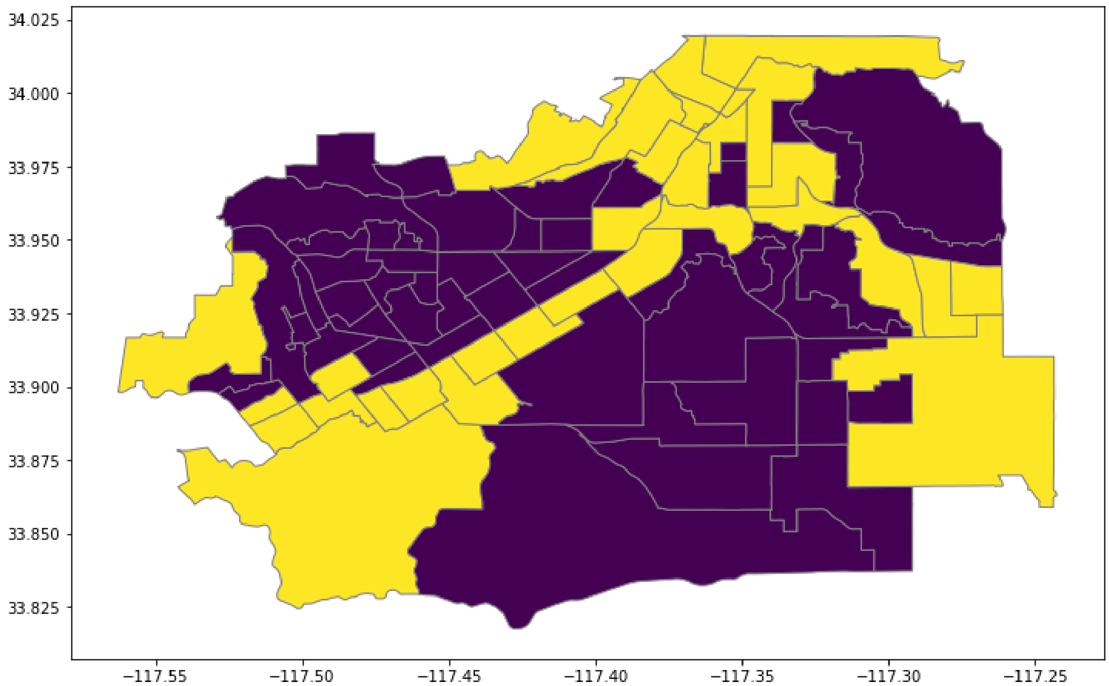
```
In [62]: city_tracts.plot(edgecolor='grey',facecolor='white')
```

```
Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36776f748>
```



Recall that previously we created the indicator variable `intersectshw` for all the tracts in Riverside County that intersected with the road network. One of the nice features of GeoPandas is that for many of the geoprocessing operations, the attributes are passed along to the derived GeoDataFrames. In our case, `city_tracts` is really just a subset of `tracts_df` so since the latter was the DataFrame that we originally defined the `intersectshw` variable, that attribute gets propagated along to the derived `city_tract` GeoDataFrame.

```
In [63]: city_tracts.plot(column='intersectshw', edgecolor='grey')

Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb367747e48>
```

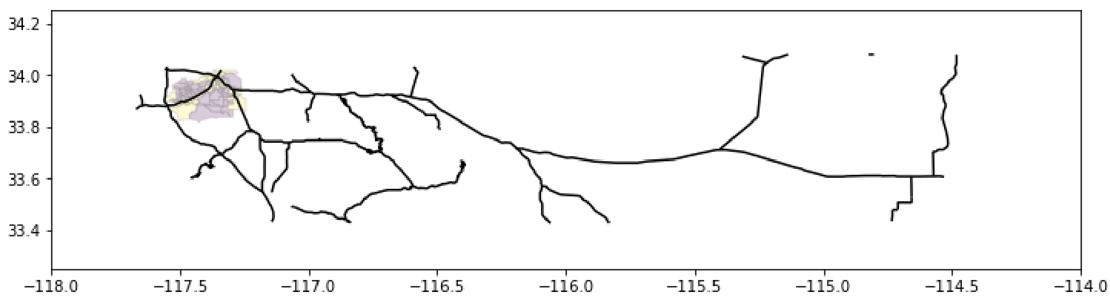


In [64]: city_tracts.head()

```
Out[64]:       GEOID10           NAMELSAD10      ALAND10  AWATER10   INTPTLAT10  \
         0  06065042012  Census Tract 420.12   2687173.0       0.0  +33.9108776
         3  06065040816  Census Tract 408.16   1788821.0       0.0  +33.9024569
         4  06065040815  Census Tract 408.15   1266779.0       0.0  +33.8930776
         5  06065040814  Census Tract 408.14   1088363.0       0.0  +33.8973552
         6  06065040813  Census Tract 408.13  12539455.0    3687.0  +33.9155438

              INTPTLON10  DP0010001  DP0010002  DP0010003  DP0010004    ...      \
         0  -117.3205065       6242        420        545        620    ...
         3  -117.5246107       2594        162        161        227    ...
         4  -117.5114997       3586        231        235        257    ...
         5  -117.5175804       4782        392        362        392    ...
         6  -117.5321545       6080        380        417        498    ...

            index_right       NAME CityType Pop2010  Land_sqmi  DateIncorp  \
         0            0  Riverside     City  303871      81.14  1883-10-11
         3            0  Riverside     City  303871      81.14  1883-10-11
         4            0  Riverside     City  303871      81.14  1883-10-11
         5            0  Riverside     City  303871      81.14  1883-10-11
         6            0  Riverside     City  303871      81.14  1883-10-11

                               WebLink     County Notes CityAbbv
         0  http://www.riversideca.gov  Riverside  None      Riv
```

```
3  http://www.riversideca.gov  Riverside   None       Riv
4  http://www.riversideca.gov  Riverside   None       Riv
5  http://www.riversideca.gov  Riverside   None       Riv
6  http://www.riversideca.gov  Riverside   None       Riv

[5 rows x 208 columns]
```

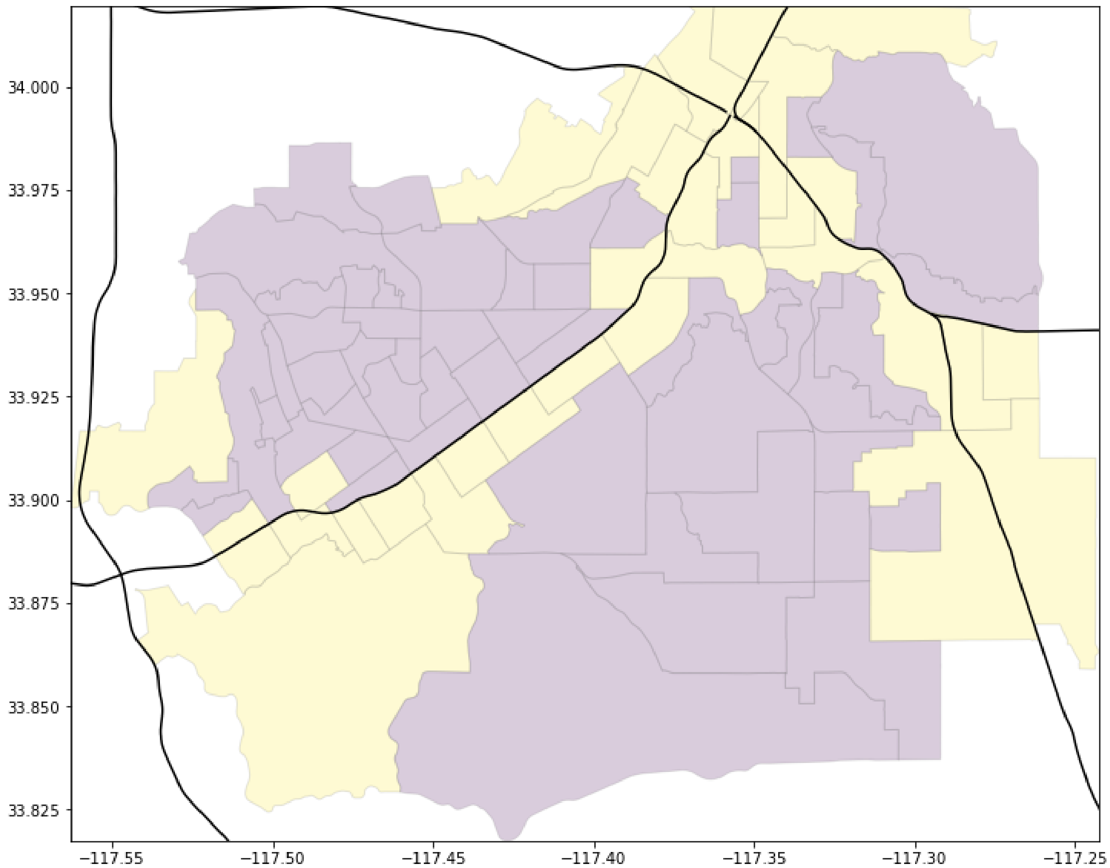
```
In [65]: plt.rcParams['figure.figsize'] = (12, 10)
         ax = plt.gca()
         city_tracts.plot(ax=ax, column='intersectshw',edgecolor='grey', alpha=0.2)
         rc_hw.plot(ax=ax, edgecolor='k')
         ax.set_xlim(-118.0, -114.0); ax.set_ylim(33.25, 34.25)
         ax.set_aspect('equal')
         plt.show()
```



Using the `total_bounds` of the new DataFrame we can zoom in to the western part of Riverside County that is centered on the City of Riverside:

```
In [66]: w, s, e, n = city_tracts.total_bounds
         w, s, e, n

Out[66]: (-117.562859, 33.81742400000013, -117.24274500000001, 34.01951400000013)
```

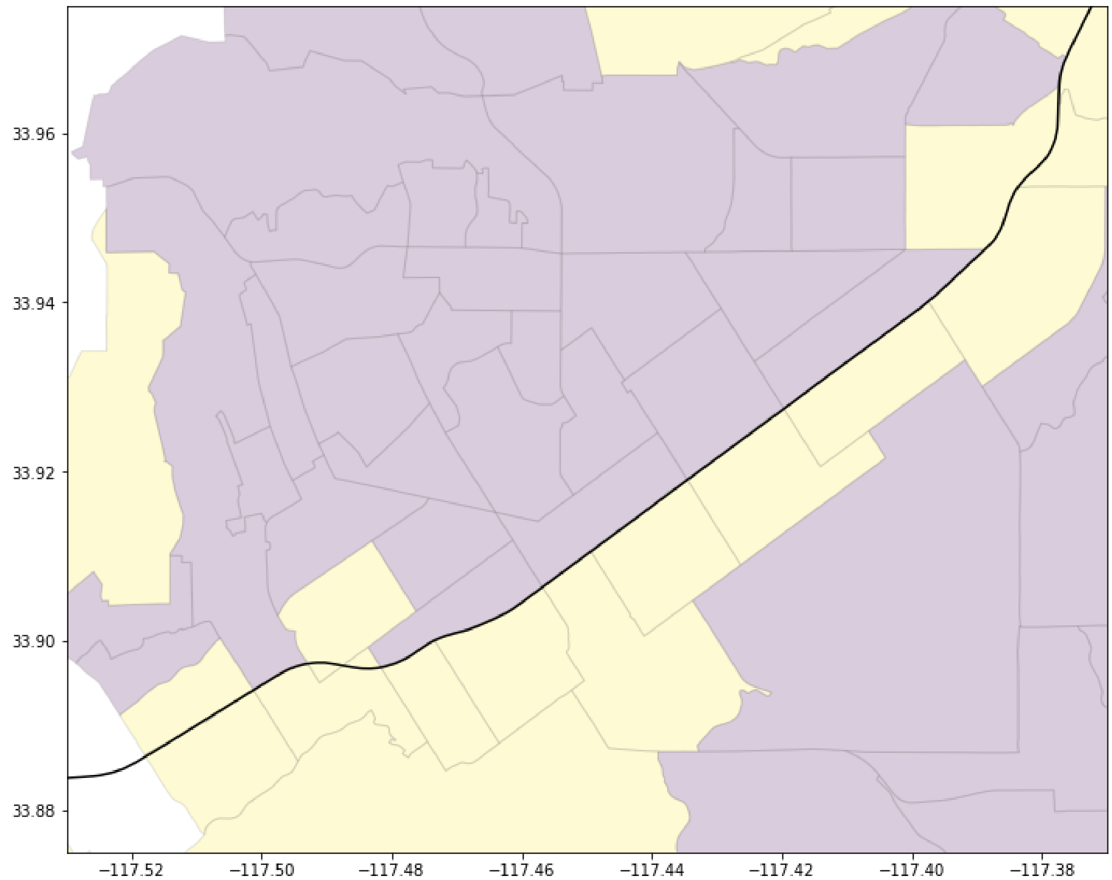
```
In [67]: plt.rcParams['figure.figsize'] = (12, 10)
         ax = plt.gca()
         city_tracts.plot(ax=ax, column='intersectshw',edgecolor='grey', alpha=0.2)
         rc_hw.plot(ax=ax, edgecolor='k')
         ax.set_xlim(w, e); ax.set_ylim(s, n)
         #ax.set_aspect('equal')
         plt.show()
```



6 Buffering

Our researcher has identified the tracts that intersect the truck route network and has sharpened the lens to the City of Riverside. However, zooming in further she sees a geographical relationship that gives her pause:

In [68]:
```
plt.rcParams['figure.figsize'] = (12, 10)
ax = plt.gca()
city_tracts.plot(ax=ax, column='intersectshw',edgecolor='grey', alpha=0.2)
rc_hw.plot(ax=ax, edgecolor='k')
ax.set_xlim(-117.53, -117.37); ax.set_ylim(33.875, 33.975)
#ax.set_aspect('equal')
plt.show()
```



It seems to her that there are cases where a segment of the road network separates two tracts, yet only one of those tracts is identified as intersecting the network. While tracts are typically defined using blocks and street center lines she would expect the tracts that share a road segment as a common part of their respective borders should both be considered intersecting the network. For her environmental equity analysis she thinks that individuals that are equidistant from the network, but on opposite sides of the highway, should face the same level of exposure. Yet, the variable she has painstakingly constructed thus far would give an asymmetric exposure measure to these individuals.

There are several reasons these apparent inconsistencies can arise. First, the origin of the tract boundaries is different from that of the route network so there is no guarantee that the same digitization process was used. Second, even if the same agency/researcher did the digitization of the two layers, if they do not follow good practice, the topological relationships may be in error. In either case, the two layers may be yield these kinds of inconsistencies when considered together.

Fortunately, our researcher knows about the concept of **buffering** and can call on this to develop a more robust representation of proximity to the highway. The idea is to define a critical distance, say 500 feet, and then define a new polygon that contains all of the points that are within 500 feed of the route network. The resulting polygon is called a **buffer**.

Once we have the 500-ft buffer, we can then repeat our intersection test for the tracts to see which tracts are within 500 feet of the route network. This would address the asymmetry problem our researcher has identified.

One issue we face, however, is that the tract CRS is unprojected:

```
In [69]: tracts_df.crs
```

```
Out[69]: {'init': 'epsg:4269'}
```

In other words, if we ignore the CRS, our distances are going to be in decimal degrees and not feet. So we need to put the tracts on a CRS with more appropriate units. Fortunately, our behavioral clinics data set has just such a CRS:

```
In [70]: clinics = gpd.read_file('data/behavioralHealth.shp')
```

```
In [71]: clinics.crs
```

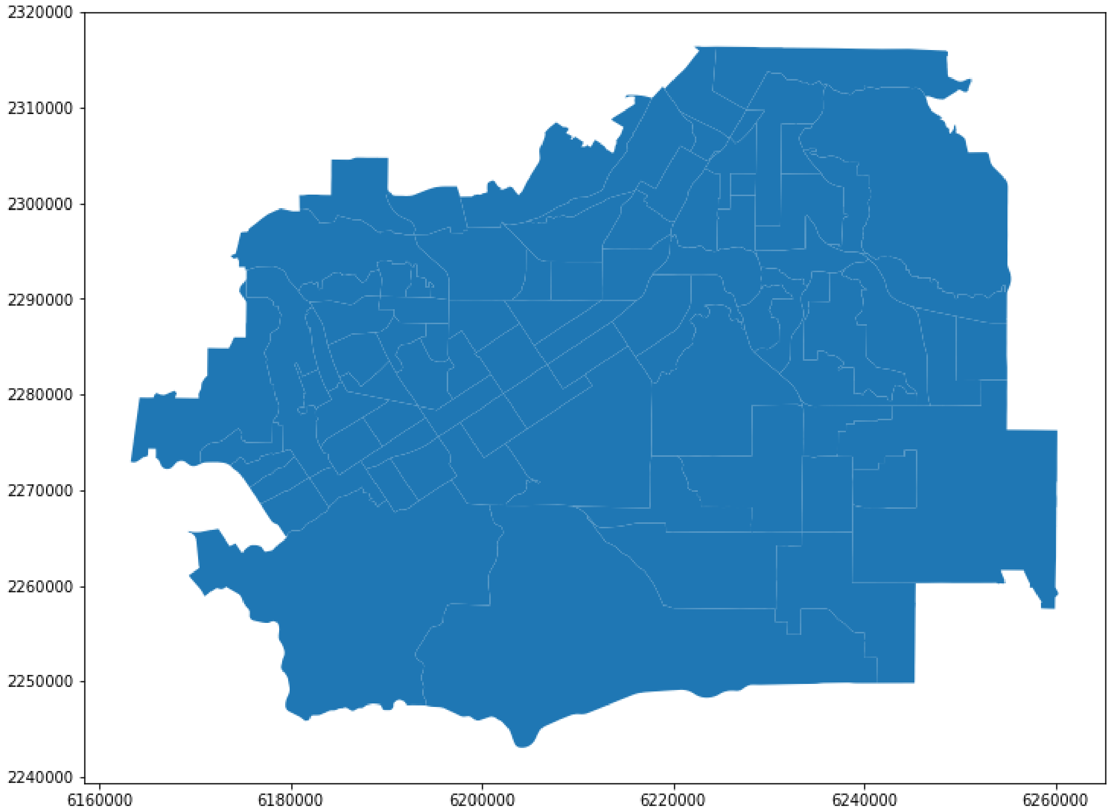
```
Out[71]: {'proj': 'lcc',
          'lat_1': 33.88333333333333,
          'lat_2': 32.78333333333333,
          'lat_0': 32.16666666666666,
          'lon_0': -116.25,
          'x_0': 2000000.000101601,
          'y_0': 500000.0001016002,
          'datum': 'NAD83',
          'units': 'us-ft',
          'no_defs': True}
```

And, we can change the CRS of the $\text{city}_{\text{tracts}}$ to that of the clinics:

```
In [72]: city_tracts = city_tracts.to_crs(clinics.crs)
```

```
In [73]: city_tracts.plot()
```

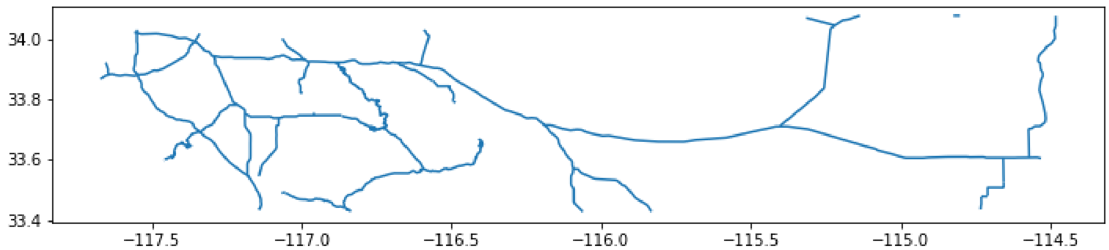
```
Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36c2d1eb8>
```



Notice that the units on the axes have changed from what we had above.
Since we will be doing a buffer around the segments of the highway in the county as well

In [74]: rc_hw.plot()

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36c2af828>



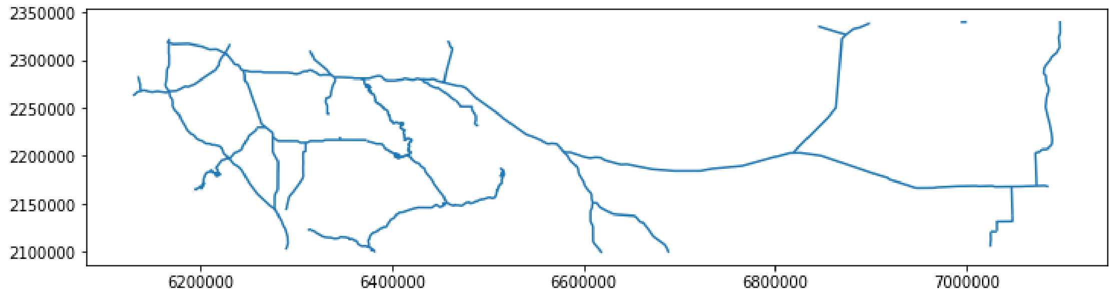
In [75]: type(rc_hw)

Out[75]: geopandas.geodataframe.GeoDataFrame

In [76]: rc_hw = rc_hw.to_crs(city_tracts.crs)

```
In [77]: rc_hw.plot()
```

```
Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36c4836d8>
```

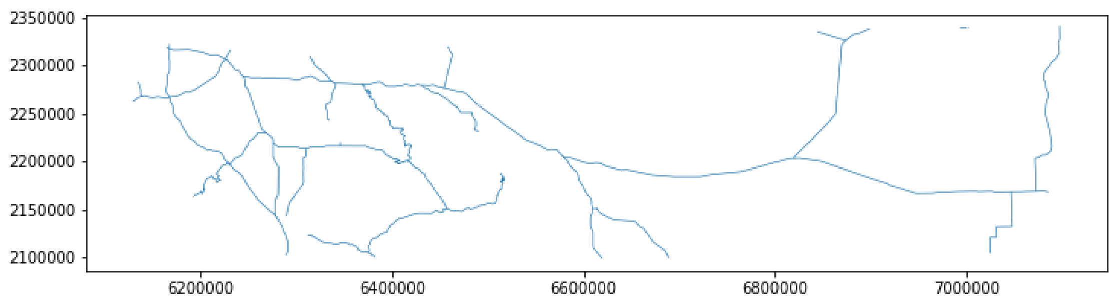


Now we can define the buffer. If we

```
In [78]: buf = rc_hw.buffer(500)
```

```
In [79]: buf.plot()
```

```
Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36c1d3e80>
```




```
In [80]: rc_hw.columns
```

```
Out[80]: Index(['geometry'], dtype='object')
```

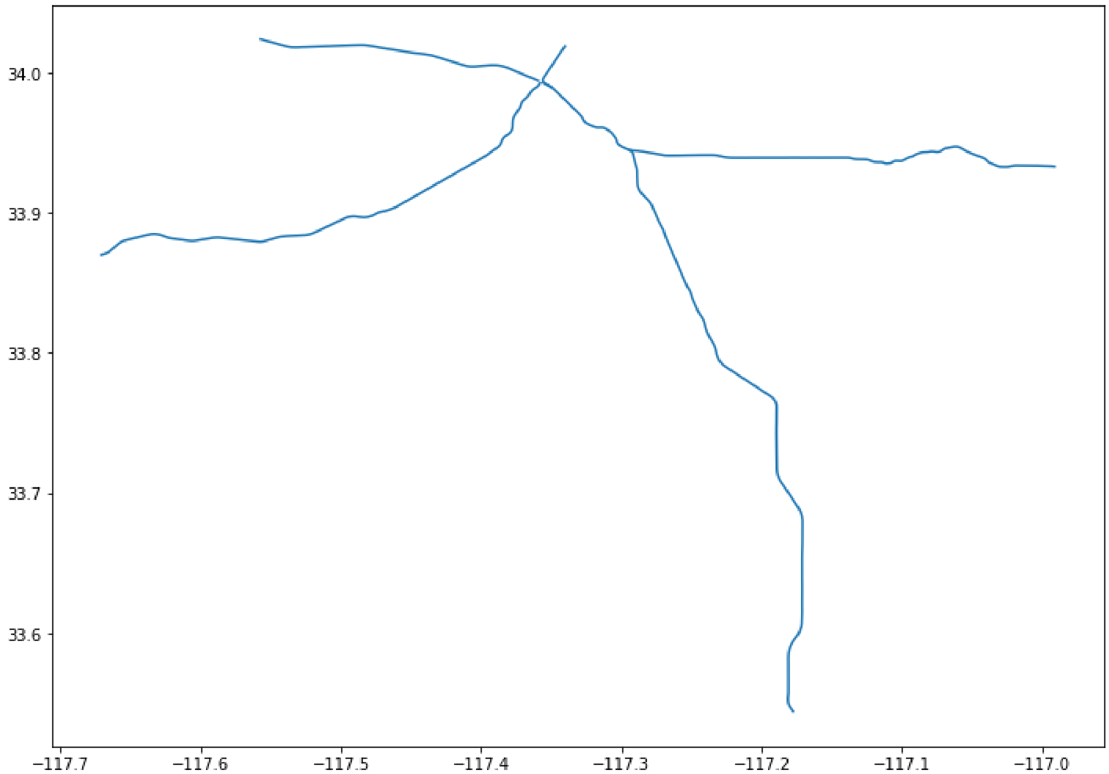
Cool. That gives us a buffer but for the network in the entire county. What about just in the city?

```
In [81]: city_tracts.columns
```

```
Out[81]: Index(['GEOID10', 'NAMELSAD10', 'ALAND10', 'AWATER10', 'INTPTLAT10',
                'INTPTLON10', 'DP0010001', 'DP0010002', 'DP0010003', 'DP0010004',
                ...
                'index_right', 'NAME', 'CityType', 'Pop2010', 'Land_sqmi', 'DateIncorp',
                'WebLink', 'County', 'Notes', 'CityAbbv'],
               dtype='object', length=208)
```

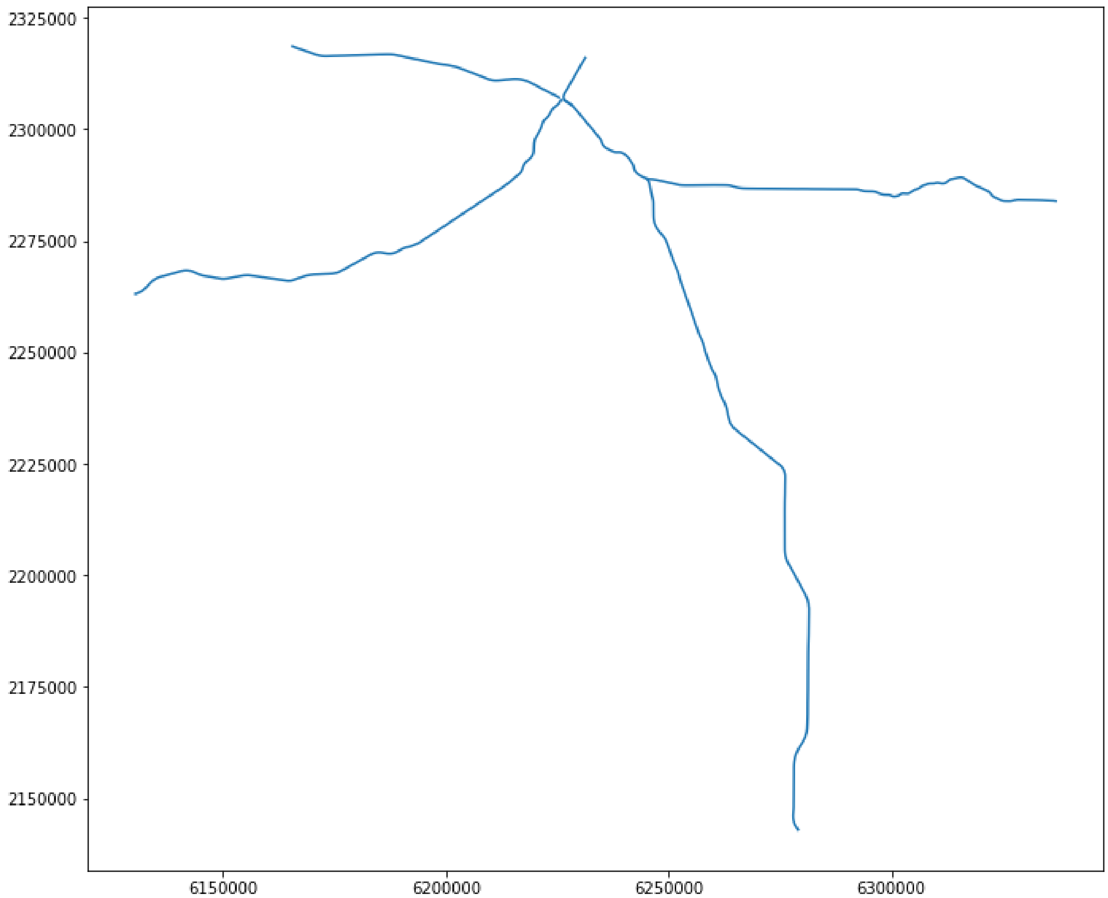
Now if we just want the segments in the city boundaries, we know a spatial join can get us these:

```
In [82]: city_hw = gpd.sjoin(routes_df, city, how='inner', op ='intersects')

In [83]: city_hw.plot()

Out[83]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36c20db38>
```



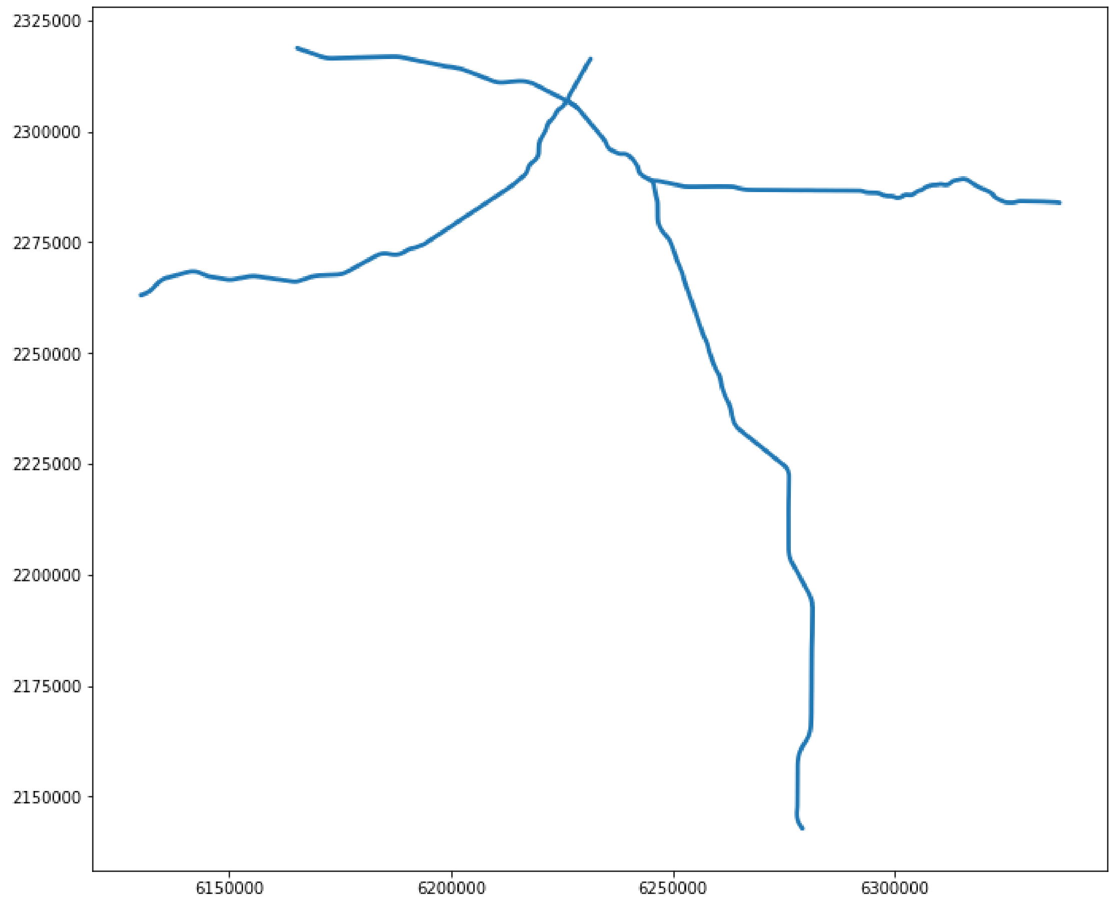
and, we take care to set its CRS accordingly:

```
In [84]: city_hw = city_hw.to_crs(city_tracts.crs)

In [85]: city_hw.plot()

Out[85]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb367676b38>
```



And, we can buffer these segments:

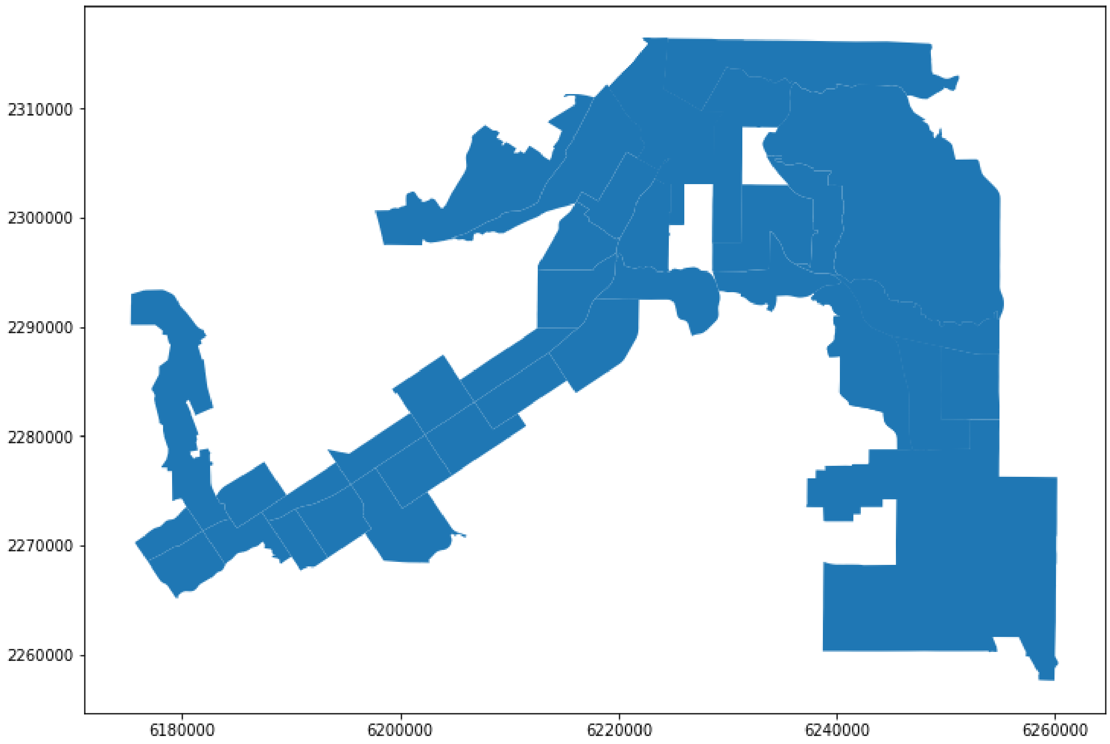
```
In [86]: b500 = city_hw.buffer(500)

In [87]: b500.plot()

Out[87]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb3675c7be0>
```




```
In [88]: ct = city_tracts[['GEOID10', 'geometry']]
         b500 = gpd.GeoDataFrame({'geometry': b500})
         b500.crs = ct.crs
```

Now we can ask to find the tracts in Riverside City that intersect with the 500-ft buffer around the highways:

```
In [89]: tracts_intersecting_hw = gpd.sjoin(ct, b500, how='inner', op='intersects')

In [90]: tracts_intersecting_hw.plot()

Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb3675a39b0>
```



This creates a new DataFrame with only those tracts for which the hit test (buffer intersection) is True:

```
In [91]: tracts_intersecting_hw.shape

Out[91]: (54, 3)
```

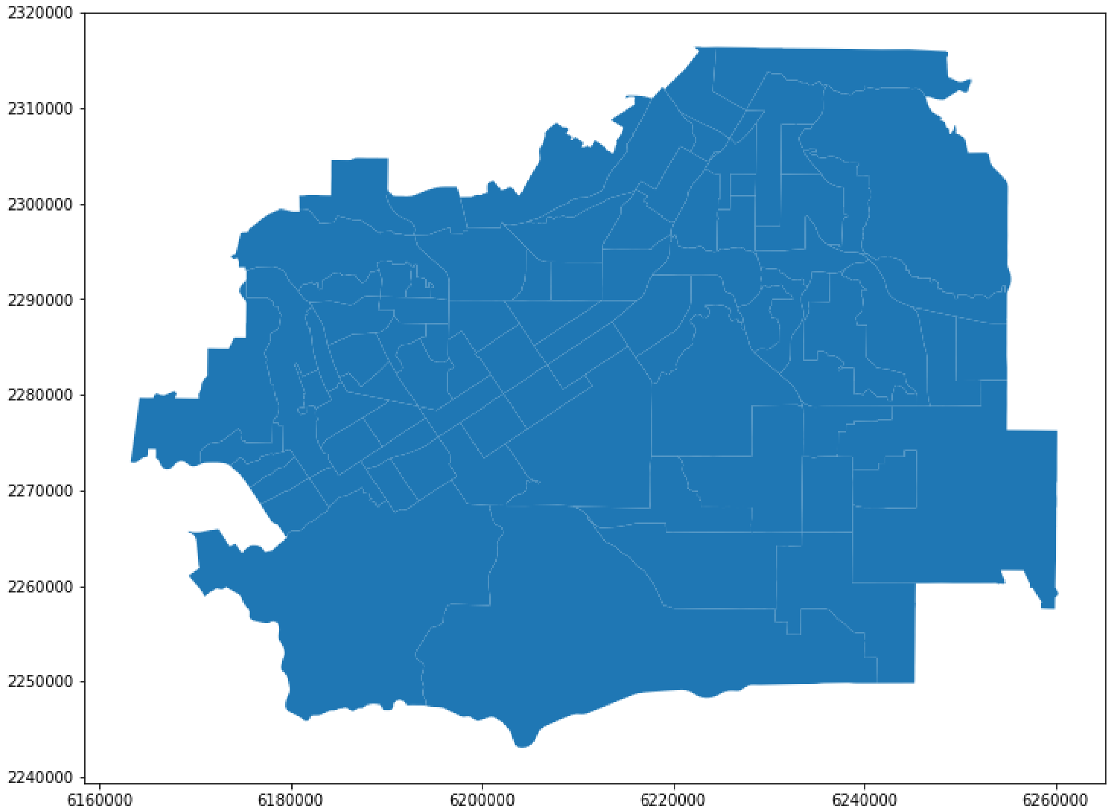
Now can create a dummy variable for these tracts to place back in the DataFrame that contains all the city tracts:

```
In [92]: geoids = city_tracts['GEOID10'].values
         tract_hw = np.array([geoid in tracts_intersecting_hw['GEOID10'].values for geoid in geo
         tract_hw

Out[92]: array([False, False,  True, False, False, False,  True, False, False,
                False,  True,  True, False,  True, False,  True, False,  True,
                 True,  True,  True,  True,  True, False, False, False,  True,
                 True,  True,  True,  True, False, False, False, False,  True,
                False, False,  True,  True, False, False, False,  True, False,
                False, False, False, False,  True, False,  True,  True,  True,
                 True, False, False,  True,  True,  True,  True, False,  True,
                 True,  True, False,  True,  True, False, False, False,  True,
                False,  True, False,  True, False, False, False, False, False,
                False, False, False])
```

```
In [93]: city_tracts['b500'] = tract_hw * 1
```

```
In [94]: city_tracts.plot()
```

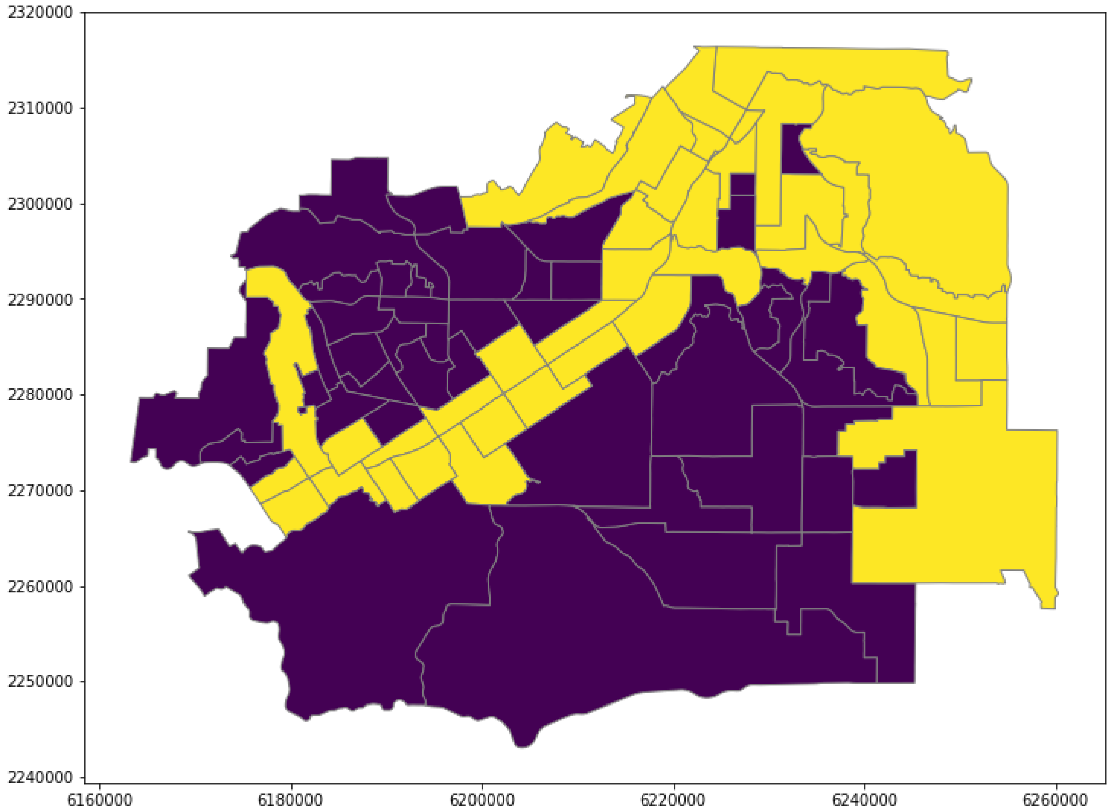
```
Out[94]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb36756c630>
```



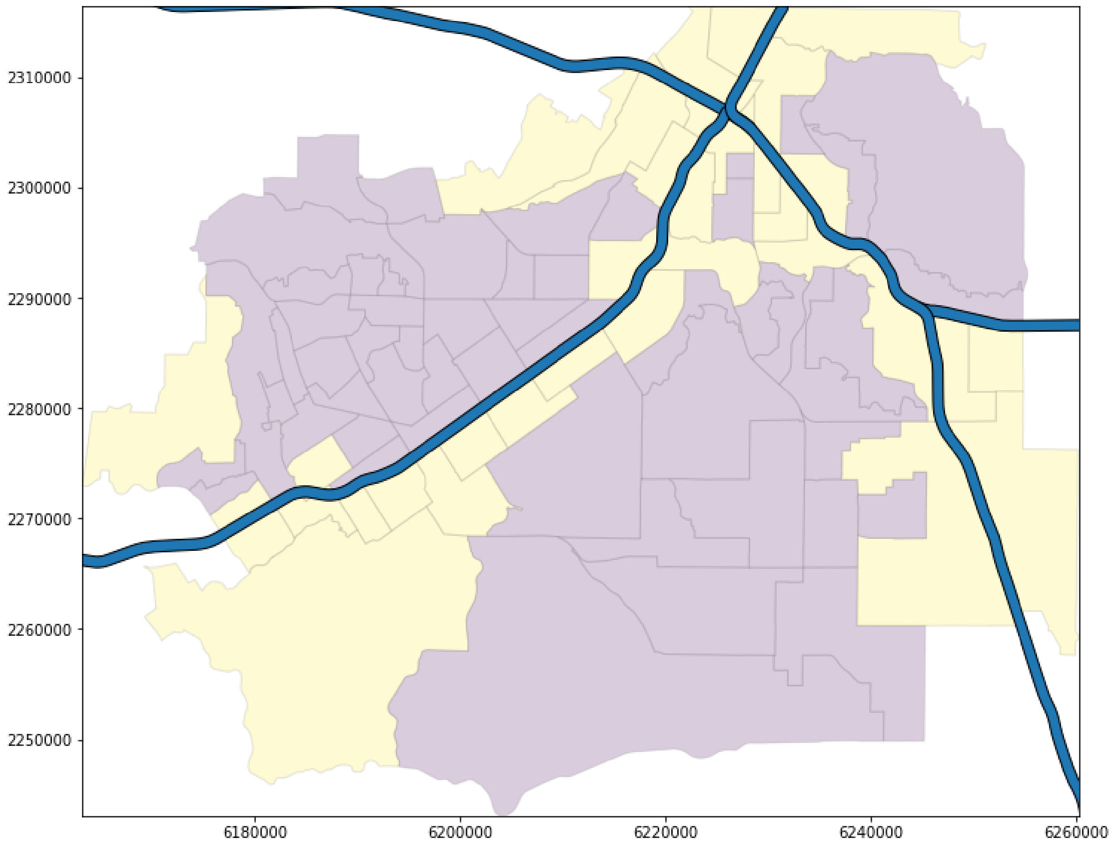
Comparing our two different operational constructs for environmental equity we have:

```
In [95]: city_tracts.plot(column='b500',edgecolor='grey')
```

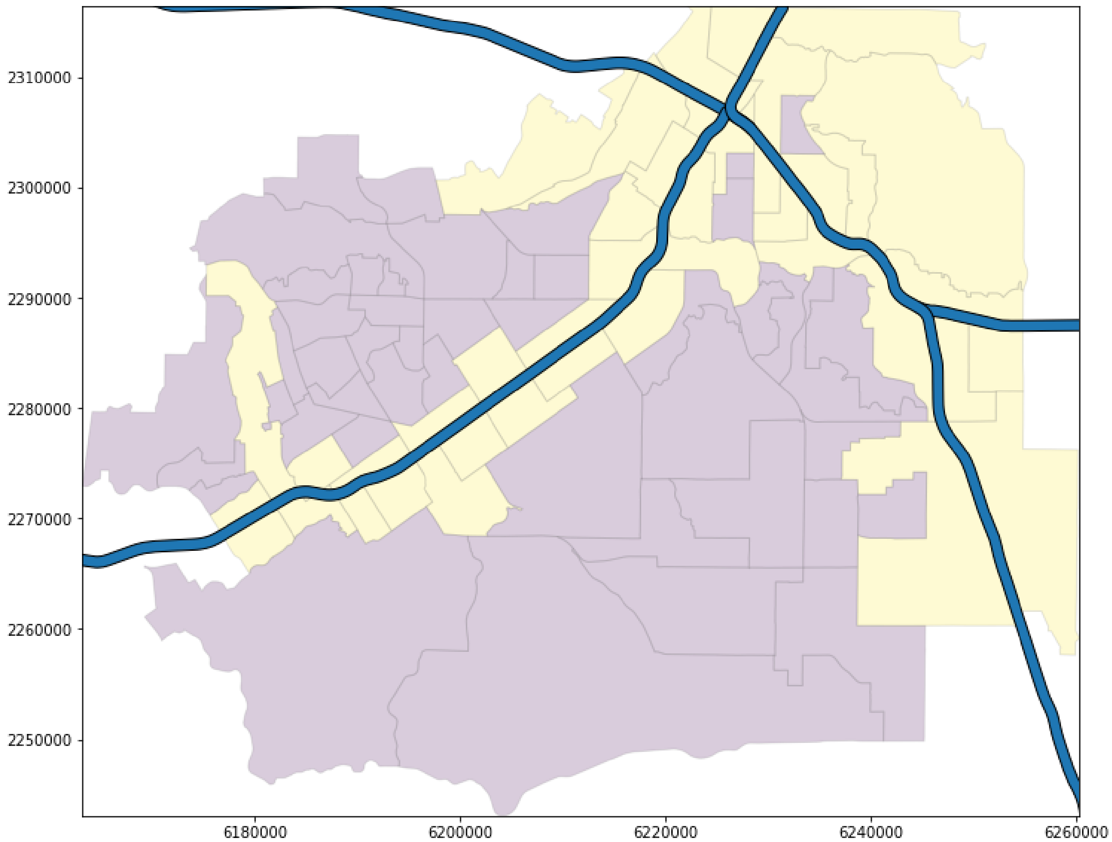
```
Out[95]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb3674c74a8>
```




```
In [96]: w, s, e, n = city_tracts.total_bounds
         plt.rcParams['figure.figsize'] = (12, 10)
         ax = plt.gca()
         city_tracts.plot(ax=ax, column='intersectshw',edgecolor='grey', alpha=0.2)
         b500.plot(ax=ax, edgecolor='k')
         ax.set_xlim(w, e); ax.set_ylim(s, n)
         #ax.set_aspect('equal')
         plt.show()
```




```
In [97]: w, s, e, n = city_tracts.total_bounds
         plt.rcParams['figure.figsize'] = (12, 10)
         ax = plt.gca()
         city_tracts.plot(ax=ax, column='b500',edgecolor='grey', alpha=0.2)
         b500.plot(ax=ax, edgecolor='k')
         ax.set_xlim(w, e); ax.set_ylim(s, n)
         #ax.set_aspect('equal')
         plt.show()
```



And we save our tracts and buffer to their own shapefiles for the next phase of our analysis.

```
In [98]: city_tracts.to_file('data/city_tracts.shp')
         b500.to_file('data/b500.shp')
```
