GeoPandas Documentation

Release 0.1.0.dev

Kelsey Jordahl

April 23, 2014

GeoPandas is an open source project to make working with geospatial data in python easier. GeoPandas extends the datatypes used by pandas to allow spatial operations on geometric types. Geometric operations are performed by shapely. Geopandas further depends on fiona for file access and descartes and matplotlib for plotting.

CHAPTER 1

Description

The goal of GeoPandas is to make working with geospatial data in python easier. It combines the capabilities of pandas and shapely, providing geospatial operations in pandas and a high-level interface to multiple geometries to shapely. GeoPandas enables you to easily do operations in python that would otherwise require a spatial database such as PostGIS.

1.1 Installation

GeoPandas is continuous-release software. You may install the latest source from GitHub and use the setup script:

```
python setup.py install
```

GeoPandas is also available on PyPI, so `pip install geopandas` should work as well. You will have to add the `--pre` flag for pip 1.4 and later.

1.1.1 Dependencies

Supports Python versions 2.6, 2.7, and 3.2+.

- numpy
- pandas
- shapely
- fiona
- six
- geopy 0.96.3 (optional; for geocoding)
- psycopg2 (optional; for PostGIS connection)

For plotting, these additional packages may be used:

- matplotlib
- descartes
- pysal

1.1.2 Testing

To run the current set of tests from the source directory, run:

```
nosetests -v
```

from a command line.

Tests are automatically run on all commits on the GitHub repository, including pull requests, on Travis CI.

1.2 GeoPandas User Guide

GeoPandas implements two main data structures, a `GeoSeries` and a `GeoDataFrame`. These are subclasses of pandas `Series` and `DataFrame`, respectively.

1.2.1 GeoSeries

A `GeoSeries` contains a sequence of geometries.

The `GeoSeries` class implements nearly all of the attributes and methods of Shapely objects. When applied to a `GeoSeries`, they will apply elementwise to all geometries in the series. Binary operations can be applied between two `GeoSeries`, in which case the operation is carried out elementwise. The two series will be aligned by matching indices. Binary operations can also be applied to a single geometry, in which case the operation is carried out for each element of the series with that geometry. In either case, a `Series` or a `GeoSeries` will be returned, as appropriate.

The following Shapely methods and attributes are available on `GeoSeries` objects:

`GeoSeries.`**`area`**
: Returns a `Series` containing the area of each geometry in the `GeoSeries`.

`GeoSeries.`**`bounds`**
: Returns a `DataFrame` with columns `minx`, `miny`, `maxx`, `maxy` values containing the bounds for each geometry. (see `GeoSeries.total_bounds` for the limits of the entire series).

`GeoSeries.`**`length`**
: Returns a `Series` containing the length of each geometry.

`GeoSeries.`**`geom_type`**
: Returns a `Series` of strings specifying the *Geometry Type* of each object.

`GeoSeries.`**`distance`**`(`*other*`)`
: Returns a `Series` containing the minimum distance to the *other* `GeoSeries` (elementwise) or geometric object.

`GeoSeries.`**`representative_point`**`()`
: Returns a `GeoSeries` of (cheaply computed) points that are guaranteed to be within each geometry.

`GeoSeries.`**`exterior`**
: Returns a `GeoSeries` of LinearRings representing the outer boundary of each polygon in the GeoSeries. (Applies to GeoSeries containing only Polygons).

`GeoSeries.`**`interiors`**
: Returns a `GeoSeries` of InteriorRingSequences representing the inner rings of each polygon in the GeoSeries. (Applies to GeoSeries containing only Polygons).

Unary Predicates

`GeoSeries.`**`is_empty`**
: Returns a `Series` of `dtype('bool')` with value `True` for empty geometries.

`GeoSeries.`**`is_ring`**
Returns a `Series` of `dtype('bool')` with value `True` for features that are closed.

`GeoSeries.`**`is_simple`**
Returns a `Series` of `dtype('bool')` with value `True` for geometries that do not cross themselves (meaningful only for *LineStrings* and *LinearRings*).

`GeoSeries.`**`is_valid`**
Returns a `Series` of `dtype('bool')` with value `True` for geometries that are valid.

Binary Predicates

`GeoSeries.`**`almost_equals`**(*other*[, *decimal=6*])
Returns a `Series` of `dtype('bool')` with value `True` if each object is approximately equal to the *other* at all points to specified *decimal* place precision. (See also `equals()`)

`GeoSeries.`**`contains`**(*other*)
Returns a `Series` of `dtype('bool')` with value `True` if each object's *interior* contains the *boundary* and *interior* of the other object and their boundaries do not touch at all.

`GeoSeries.`**`crosses`**(*other*)
Returns a `Series` of `dtype('bool')` with value `True` if the *interior* of each 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.

`GeoSeries.`**`disjoint`**(*other*)
Returns a `Series` of `dtype('bool')` with value `True` if the *boundary* and *interior* of each object does not intersect at all with those of the other.

`GeoSeries.`**`equals`**(*other*)
Returns a `Series` of `dtype('bool')` with value `True` if if the set-theoretic *boundary*, *interior*, and *exterior* of each object coincides with those of the other.

`GeoSeries.`**`intersects`**(*other*)
Returns a `Series` of `dtype('bool')` with value `True` if if the *boundary* and *interior* of each object intersects in any way with those of the other.

`GeoSeries.`**`touches`**(*other*)
Returns a `Series` of `dtype('bool')` with value `True` if the objects have at least one point in common and their interiors do not intersect with any part of the other.

`GeoSeries.`**`within`**(*other*)
Returns a `Series` of `dtype('bool')` with value `True` if each object's *boundary* and *interior* intersect only with the *interior* of the other (not its *boundary* or *exterior*). (Inverse of `contains()`)

Set-theoretic Methods

`GeoSeries.`**`boundary`**
Returns a `GeoSeries` of lower dimensional objects representing each geometries's set-theoretic *boundary*.

`GeoSeries.`**`centroid`**
Returns a `GeoSeries` of points for each geometric centroid.

`GeoSeries.`**`difference`**(*other*)
Returns a `GeoSeries` of the points in each geometry that are not in the *other* object.

`GeoSeries.`**`intersection`**(*other*)
Returns a `GeoSeries` of the intersection of each object with the *other* geometric object.

`GeoSeries.`**`symmetric_difference`**(*other*)
Returns a `GeoSeries` of the points in each object not in the *other* geometric object, and the points in the *other* not in this object.

GeoSeries.**union**(*other*)
Returns a GeoSeries of the union of points from each object and the *other* geometric object.

Constructive Methods

GeoSeries.**buffer**(*distance*, *resolution=16*)
Returns a GeoSeries of geometries representing all points within a given *distance* of each geometric object.

GeoSeries.**convex_hull**
Returns a GeoSeries of geometries representing the smallest convex *Polygon* containing all the points in each object unless the number of points in the object is less than three. For two points, the convex hull collapses to a *LineString*; for 1, a *Point*.

GeoSeries.**envelope**
Returns a GeoSeries of geometries representing the point or smallest rectangular polygon (with sides parallel to the coordinate axes) that contains each object.

GeoSeries.**simplify**(*tolerance*, *preserve_topology=True*)
Returns a GeoSeries containing a simplified representation of each object.

Affine transformations

GeoSeries.**rotate**(*self*, *angle*, *origin='center'*, *use_radians=False*)
Rotate the coordinates of the GeoSeries.

GeoSeries.**scale**(*self*, *xfact=1.0*, *yfact=1.0*, *zfact=1.0*, *origin='center'*)
Scale the geometries of the GeoSeries along each (x, y, z) dimensio.

GeoSeries.**skew**(*self*, *angle*, *origin='center'*, *use_radians=False*)
Shear/Skew the geometries of the GeoSeries by angles along x and y dimensions.

GeoSeries.**translate**(*self*, *angle*, *origin='center'*, *use_radians=False*)
Shift the coordinates of the GeoSeries.

Aggregating methods

GeoSeries.**unary_union**
Return a geometry containing the union of all geometries in the GeoSeries.

Additionally, the following methods are implemented:

GeoSeries.**from_file**()
Load a GeoSeries from a file from any format recognized by fiona.

GeoSeries.**to_crs**(*crs=None*, *epsg=None*)
Transform all geometries in a GeoSeries to a different coordinate reference system. The crs attribute on the current GeoSeries must be set. Either crs in dictionary form or an EPSG code may be specified for output.

This method will transform all points in all objects. It has no notion or projecting entire geometries. All segments joining points are assumed to be lines in the current projection, not geodesics. Objects crossing the dateline (or other projection boundary) will have undesirable behavior.

GeoSeries.**plot**(*colormap='Set1'*, *alpha=0.5*, *axes=None*)
Generate a plot of the geometries in the GeoSeries. colormap can be any recognized by matplotlib, but discrete colormaps such as Accent, Dark2, Paired, Pastel1, Pastel2, Set1, Set2, or Set3 are recommended. Wraps the plot_series() function.

GeoSeries.**total_bounds**
Returns a tuple containing minx, miny, maxx, maxy values for the bounds of the series as a whole. See GeoSeries.bounds for the bounds of the geometries contained in the series.

Methods of pandas `Series` objects are also available, although not all are applicable to geometric objects and some may return a `Series` rather than a `GeoSeries` result. The methods `copy()`, `align()`, `isnull()` and `fillna()` have been implemented specifically for `GeoSeries` and are expected to work correctly.

1.2.2 GeoDataFrame

A `GeoDataFrame` is a tablular data structure that contains a column called `geometry` which contains a *GeoSeries'*.

Currently, the following methods are implemented for a `GeoDataFrame`:

classmethod `GeoDataFrame.`**`from_file`**(*filename*, ***kwargs*)
Load a `GeoDataFrame` from a file from any format recognized by fiona. See `read_file()`.

classmethod `GeoDataFrame.`**`from_postgis`**(*sql*, *con*, *geom_col='geom'*, *crs=None*, *index_col=None*, *coerce_float=True*, *params=None*)
Load a `GeoDataFrame` from a file from a PostGIS database. See `read_postgis()`.

`GeoSeries.`**`to_crs`**(*crs=None*, *epsg=None*, *inplace=False*)
Transform all geometries in the `geometry` column of a GeoDataFrame to a different coordinate reference system. The `crs` attribute on the current GeoSeries must be set. Either `crs` in dictionary form or an EPSG code may be specified for output. If `inplace=True` the geometry column will be replaced in the current dataframe, otherwise a new GeoDataFrame will be returned.

This method will transform all points in all objects. It has no notion or projecting entire geometries. All segments joining points are assumed to be lines in the current projection, not geodesics. Objects crossing the dateline (or other projection boundary) will have undesirable behavior.

`GeoSeries.`**`to_file`**(*filename*, *driver="ESRI Shapefile"*, ***kwargs*)
Write the `GeoDataFrame` to a file. By default, an ESRI shapefile is written, but any OGR data source supported by Fiona can be written. `**kwargs` are passed to the Fiona driver.

`GeoSeries.`**`to_json`**(***kwargs*)
Returns a GeoJSON representation of the `GeoDataFrame` as a string.

`GeoDataFrame.`**`plot`**(*column=None*, *colormap=None*, *alpha=0.5*, *categorical=False*, *legend=False*, *axes=None*)
Generate a plot of the geometries in the `GeoDataFrame`. If the `column` parameter is given, colors plot according to values in that column, otherwise calls `GeoSeries.plot()` on the `geometry` column. Wraps the `plot_dataframe()` function.

All pandas `DataFrame` methods are also available, although they may not operate in a meaningful way on the `geometry` column and may not return a `GeoDataFrame` result even when it would be appropriate to do so.

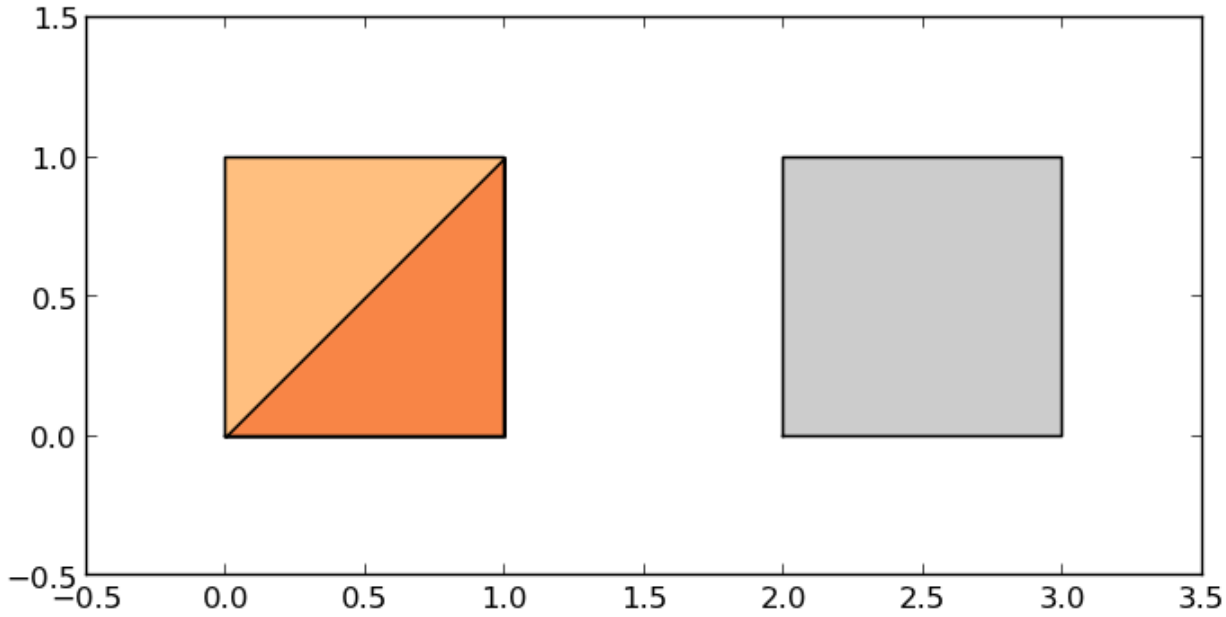
1.2.3 Geopandas functions

`geopandas.geocode.`**`geocode`**(*strings*, *provider='googlev3'*, ***kwargs*)
Geocode a list of strings and return a GeoDataFrame containing the resulting points in its `geometry` column. Available `provider``s include ``googlev3`, `bing`, `google`, `yahoo`, `mapquest`, and `openmapquest`. `**kwargs` will be passed as parameters to the appropriate geocoder.

Requires geopy. Please consult the Terms of Service for the chosen provider.

1.2.4 Examples

```
>>> p1 = Polygon([(0, 0), (1, 0), (1, 1)])
>>> p2 = Polygon([(0, 0), (1, 0), (1, 1), (0, 1)])
>>> p3 = Polygon([(2, 0), (3, 0), (3, 1), (2, 1)])
>>> g = GeoSeries([p1, p2, p3])
>>> g
0    POLYGON ((0.0000000000000000 0.000000000000000...
1    POLYGON ((0.0000000000000000 0.000000000000000...
2    POLYGON ((2.0000000000000000 0.000000000000000...
dtype: object
```

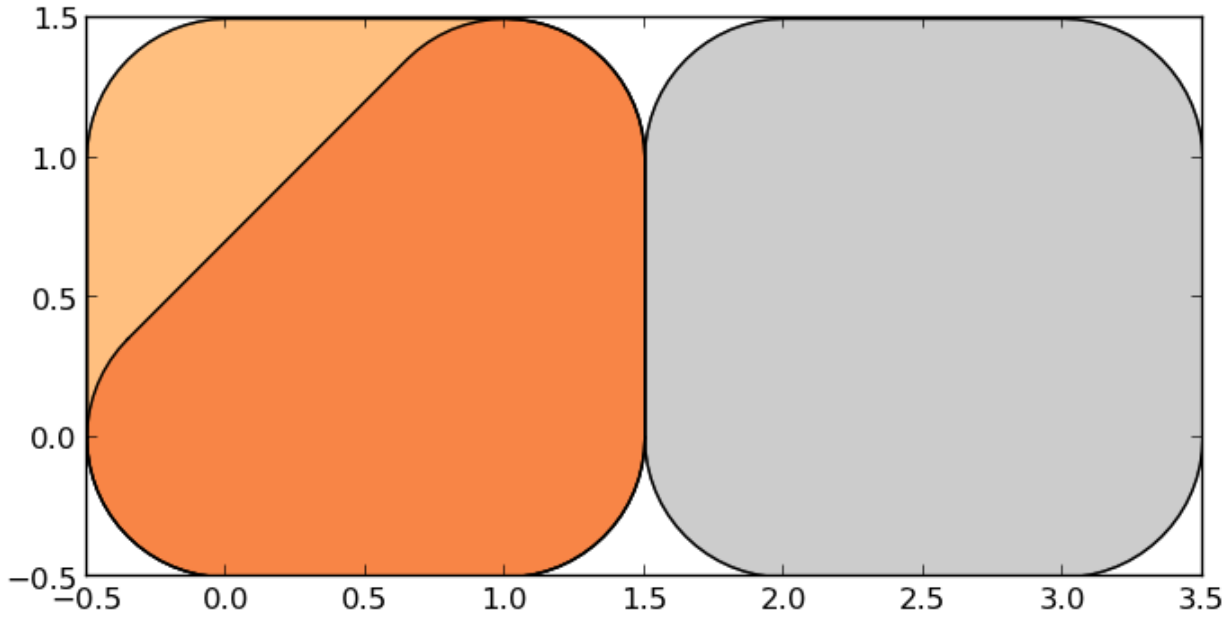


Some geographic operations return normal pandas object. The `area` property of a `GeoSeries` will return a `pandas.Series` containing the area of each item in the `GeoSeries`:

```
>>> print g.area
0    0.5
1    1.0
2    1.0
dtype: float64
```

Other operations return GeoPandas objects:

```
>>> g.buffer(0.5)
Out[15]:
0    POLYGON ((-0.3535533905932737 0.35355339059327...
1    POLYGON ((-0.5000000000000000 0.00000000000000...
2    POLYGON ((1.5000000000000000 0.000000000000000...
dtype: object
```



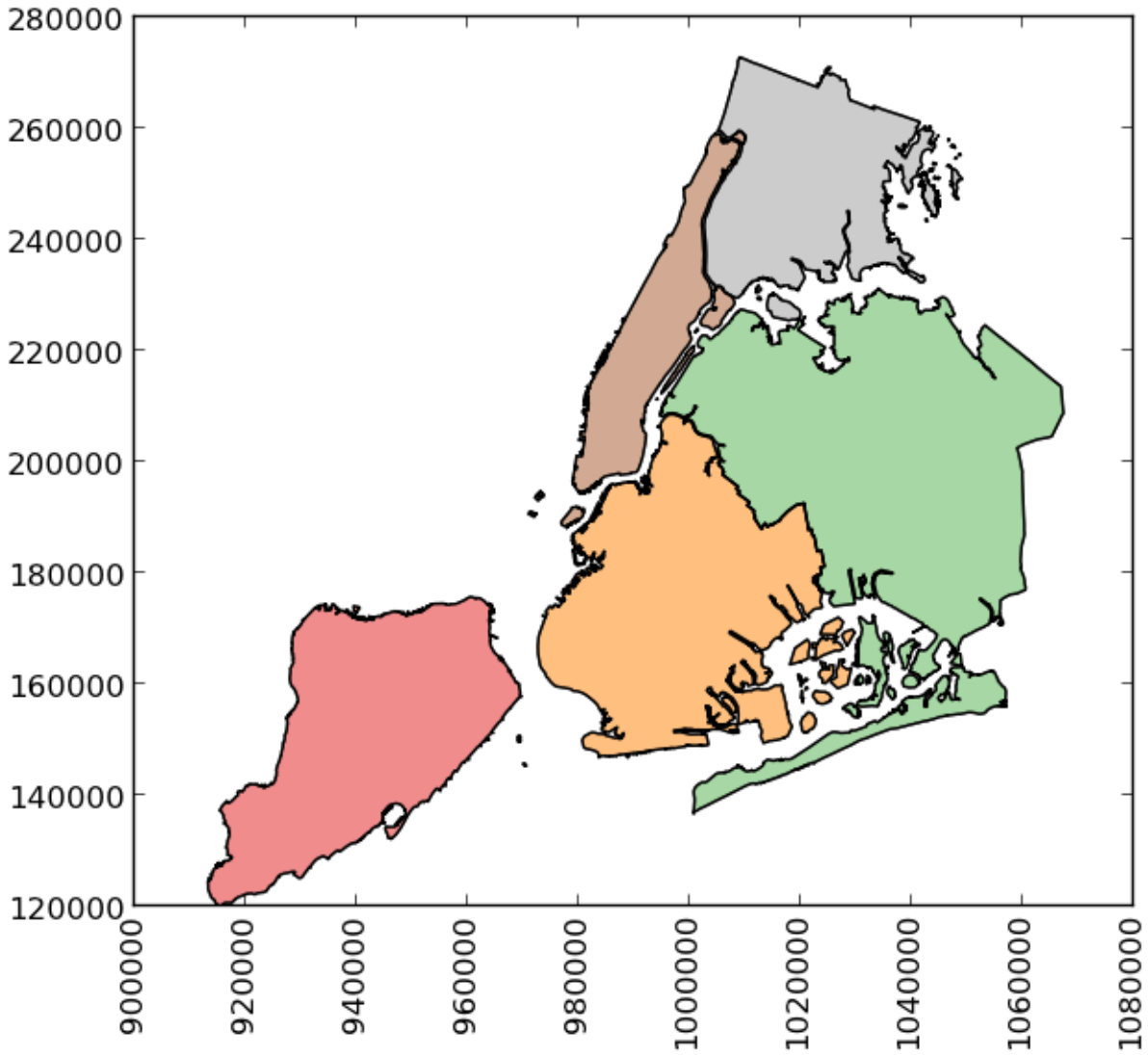
GeoPandas objects also know how to plot themselves. GeoPandas uses descartes to generate a matplotlib plot. To generate a plot of our GeoSeries, use:

```
>>> g.plot()
```

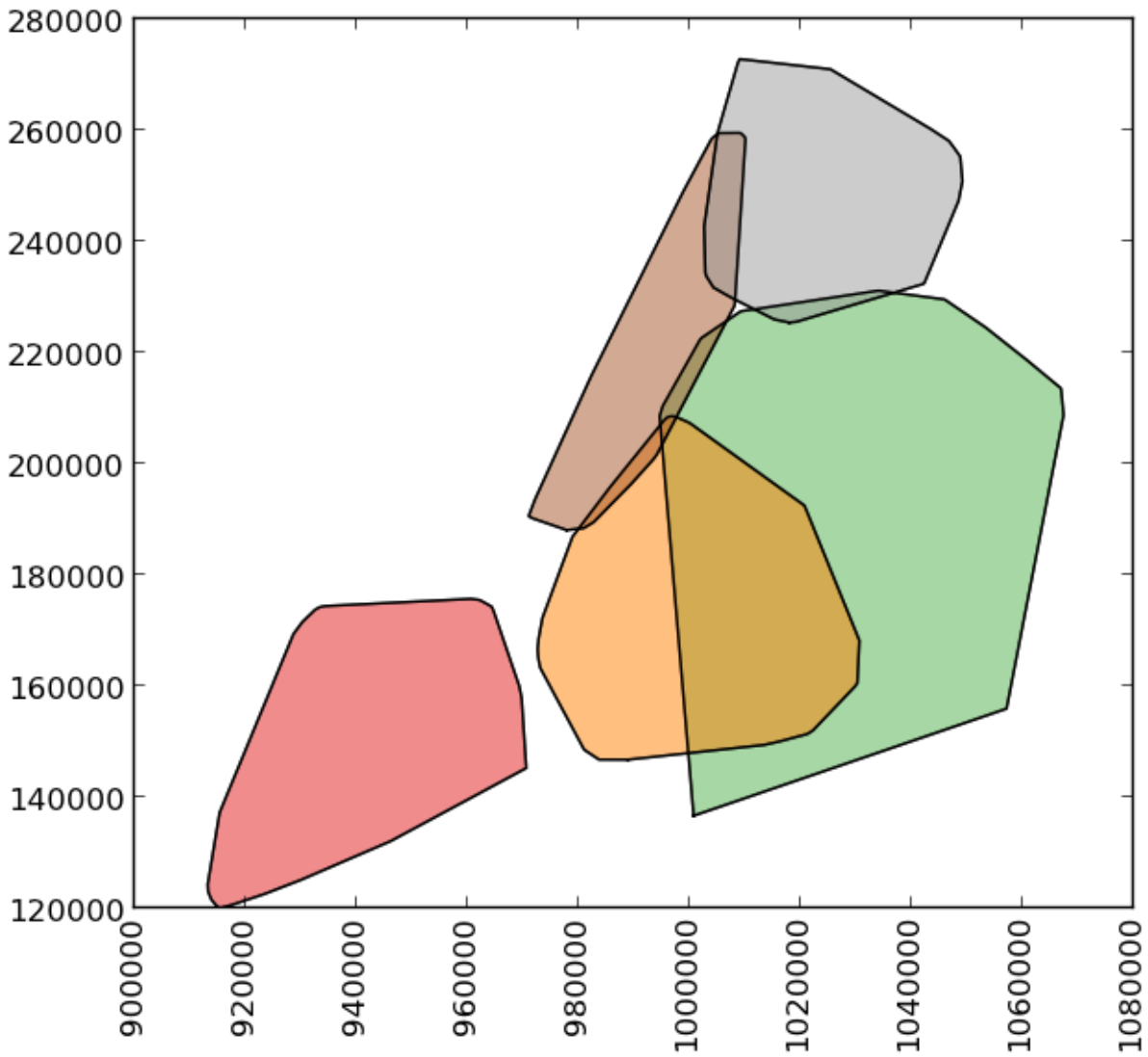
GeoPandas also implements alternate constructors that can read any data format recognized by fiona. To read a file containing the boroughs of New York City:

```
>>> boros = GeoDataFrame.from_file('nybb.shp')
>>> boros.set_index('BoroCode', inplace=True)
>>> boros.sort()
>>> boros
               BoroName    Shape_Area     Shape_Leng  \
BoroCode
1             Manhattan  6.364422e+08  358532.956418
2                 Bronx  1.186804e+09  464517.890553
3              Brooklyn  1.959432e+09  726568.946340
4                Queens  3.049947e+09  861038.479299
5         Staten Island  1.623853e+09  330385.036974

                                                  geometry
BoroCode
1         (POLYGON ((981219.0557861328125000 188655.3157...
2         (POLYGON ((1012821.8057861328125000 229228.264...
3         (POLYGON ((1021176.4790039062500000 151374.796...
4         (POLYGON ((1029606.0765991210937500 156073.814...
5         (POLYGON ((970217.0223999023437500 145643.3322...
```




```
>>> boros['geometry'].convex_hull
0    POLYGON ((915517.6877458114176989 120121.88125...
1    POLYGON ((1000721.5317993164062500 136681.7761...
2    POLYGON ((988872.8212280273437500 146772.03179...
3    POLYGON ((977855.4451904296875000 188082.32238...
4    POLYGON ((1017949.9776000976562500 225426.8845...
dtype: object
```



To demonstrate a more complex operation, we'll generate a `GeoSeries` containing 2000 random points:

```
>>> from shapely.geometry import Point
>>> xmin, xmax, ymin, ymax = 900000, 1080000, 120000, 280000
>>> xc = (xmax - xmin) * np.random.random(2000) + xmin
>>> yc = (ymax - ymin) * np.random.random(2000) + ymin
>>> pts = GeoSeries([Point(x, y) for x, y in zip(xc, yc)])
```

Now draw a circle with fixed radius around each point:

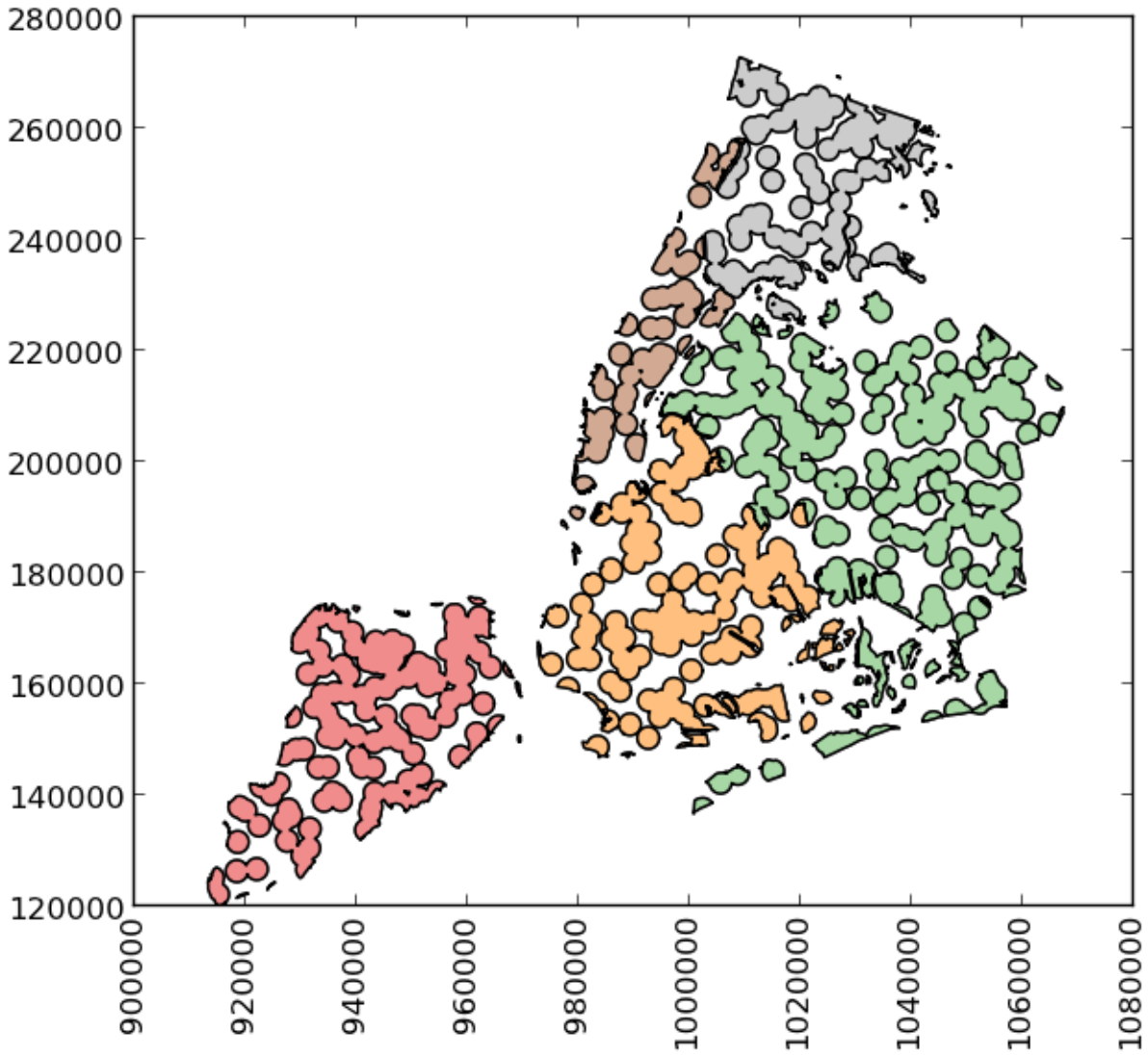
```
>>> circles = pts.buffer(2000)
```

We can collapse these circles into a single shapely MultiPolygon geometry with

```
>>> mp = circles.unary_union
```

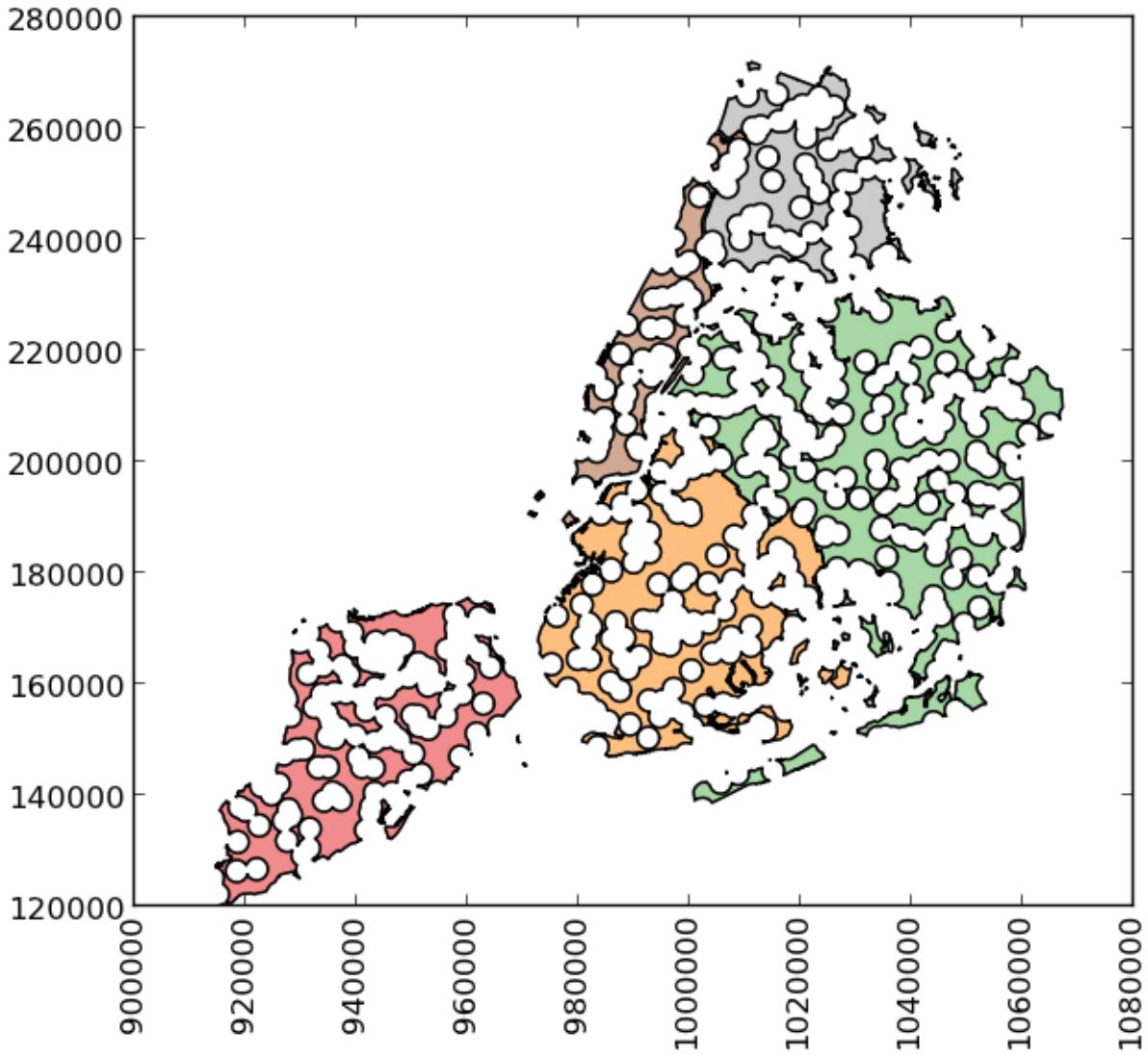
To extract the part of this geometry contained in each borough, we can just use:

```
>>> holes = boros['geometry'].intersection(mp)
```



and to get the area outside of the holes:

```
>>> boros_with_holes = boros['geometry'].difference(mp)
```



Note that this can be simplified a bit, since `geometry` is available as an attribute on a `GeoDataFrame`, and the `intersection` and `difference` methods are implemented with the "&" and "-" operators, respectively. For example, the latter could have been expressed simply as `boros.geometry - mp`.

It's easy to do things like calculate the fractional area in each borough that are in the holes:

```
>>> holes.area / boros.geometry.area
BoroCode
1           0.602015
2           0.523457
3           0.585901
4           0.577020
5           0.559507
dtype: float64
```

1.3 About

Coming soon...

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*