Interaktive Visualisierung von Geodaten in Jupyter Notebooks (Lightning Talk, FOSSGIS 2017)

March 23, 2017

1 Shapely in Jupyter Notebook ist total toll!

Johannes Kröger | HafenCity Universität Hamburg

johannes.kroeger (ätt) hcu-hamburg.de | [@cartocalypse](https://www.twitter.com/cartocalypse)

Mein Code ist teilweise absolut nicht hübsch und "there must be a better way" dürfte oftmals zutreffen. Aber darum geht es in diesem Notebook ja auch nicht. :o)

1.1 Datenquellen

- **hamburg-latest-free_buildings_20170323_25832.shp**: http://download.geofabrik.de/europe/germany/h latest-free.shp.zip vom 2017-03-23, Data Copyright OpenStreetMap Contributors in QGIS nach EPSG:25832 projiziert und sämtliche Attribute entfernt.
- **ne_10m_admin_0_countries_germany25832.shp**: Deutschland aus http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/10m/cultural/ne_10m geholt und nach UTM32N projiziert. Public Domain.
- **ne_110m_admin_0_countries.shp**: http://www.naturalearthdata.com/http//www.naturalearthdata.com Public Domain.
- **Strassenbaumkataster_HH_2017-01-06.shp**: http://daten-hamburg.de/umwelt_klima/strassenbaumkata 01-06_GML.zip, mit QGIS Attribute entfernt und als Shapefile gespeichert. Freie und Hansestadt Hamburg, Behörde für Umwelt und Energie, Datenlizenz Deutschland Namensnennung 2.0

1.2 Los geht's

Graue Zellen sind Code, alles andere entweder Text von mir oder die Ausgabe des Codes.

Erstmal eine schlichte Python-Funktion, die die übergebene Zahl quadriert und das Ergebnis zurück gibt.

```
In [1]: def square(x):
            return x*x

In [2]: print(square(3))
```

```
9
```

In Jupyter Notebook werden Rückgabewerte direkt angezeigt, ähnlich wie in einem interaktiven Interpreter:

```
In [3]: square(3)

Out[3]: 9
```

Mit den IPython widgets kann man tolle interaktive Sachen machen. Mit `interactive` bekommt man interaktive "Widgets" für die Funktionsparameter:

```
In [4]: from ipywidgets import interactive
```

Man übergibt die Funktion und mögliche Werte für die Eingabeparameter. `interactive` baut daraus total smart ein interaktives Widget und ruft die Funktion mit dem entsprechenden Wert(en) auf.

```
In [5]: interactive(square, x=10)
```

Übergibt man ein Tuple, dann baut `interactive` eine sinnvolle "Range" daraus ala `(start [, stop] [, increment])`.

```
In [6]: interactive(square, x=(0,100,2))
```

Übergibt man eine Liste, dann bekommt man ein Dropdown mit den Elementen:

```
In [7]: interactive(square, x=range(0,100))
```

Man kann natürlich auch mehrere Variablen entsprechend manipulieren

```
In [8]: def power(x, exponent):
            return x**exponent

In [9]: interactive(power, x=(0,10), exponent=(0,5))

In [10]: interactive(power, x=(0,10), exponent=2)
```

Will man einen Parameter übergeben, aber nicht interaktiv beeinfluSSen können, so bietet sich fixed() an (oder man benutzt globale Variablen und genieSSt das Knistern):

```
In [11]: from ipywidgets import fixed

In [12]: interactive(power, x=(0,10), exponent=fixed(2))
```

Toll oder? But let's even take it up a notch!

1.3 Shapely rendert seine Geometrieobjekte grafisch inline in Jupyter Notebooks ø/

Shapely ist "die" Geometriebibliothek für Python. Das allseits geliebte JTS/GEOS halt, hier pythonisiert.

```
In [13]: from shapely.geometry import Point

In [14]: p1 = Point(0, 0)
```

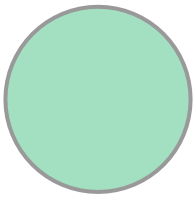
Weil Shapely so toll ist, werden seine Geometrieobjekte in Jupyter Notebooks als SVG gerendert:

```
In [15]: p1
```

Out[15]:


```
In [16]: b1 = p1.buffer(5)
         b1
```

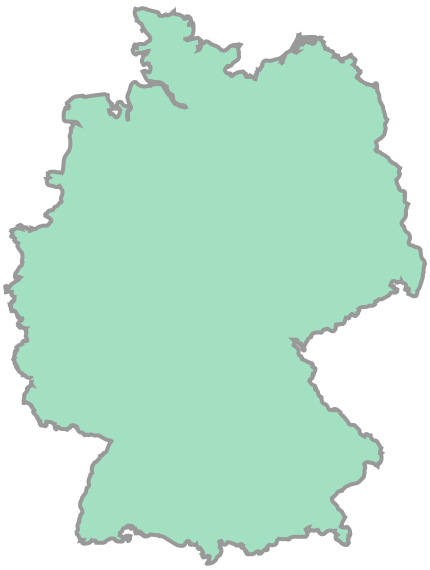
Out[16]:



Gleich wird's auch hiermit interaktiv, aber lasst uns erstmal eine nettere Geometrie benutzen.

```
In [17]: import fiona
         from shapely.geometry import shape
         with fiona.open("ne_10m_admin_0_countries_germany25832.shp", 'r') as source:
             germany = shape(source[0]['geometry'])
         germany = sorted(germany, key=lambda g: -g.area)[0]  # nur das größte Polygon, damit wi
         germany
```

Out[17]:



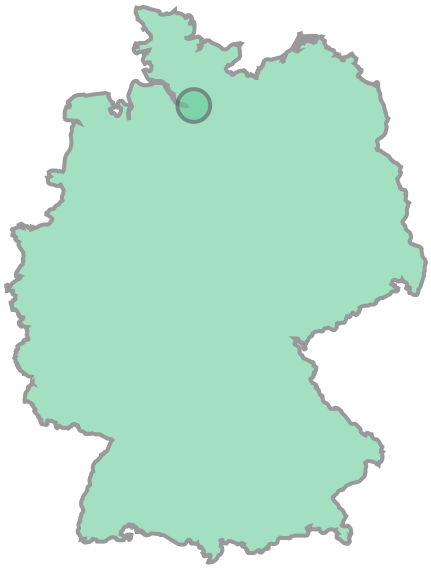
```
In [18]: def buffer(g, radius):
             # No idea if you could just pass instance methods to interactive() instead 8)
             return g.buffer(radius)

In [19]: from ipywidgets import fixed  # wenn ein Parameter fix ist und kein Widget dafür erzeug
         interactive(buffer, g=fixed(germany), radius=(-20000,20000,2500))

In [20]: from shapely.geometry import box, GeometryCollection

In [21]: from shapely.affinity import translate

         def moveme(g, xoffset, yoffset):
             frame = box(*g.bounds).exterior
             translated = translate(g, xoff=xoffset, yoff=yoffset)
             return GeometryCollection((frame, translated))

         germany_width = int(germany.bounds[2]-germany.bounds[0])
         germany_height = int(germany.bounds[3]-germany.bounds[1])
         interactive(
             moveme,
             g=fixed(germany),
             xoffset=(-germany_width-1,germany_width+1,germany_width/10),
             yoffset=(-germany_height-1,germany_height+1,germany_height/10)
         )

In [22]: from shapely.affinity import rotate

         def rotateme(g, angle):
```

```
        frame = box(*g.bounds).exterior
        rotated = rotate(g, angle=angle)
        return GeometryCollection((frame, rotated))

    interactive(rotateme, g=fixed(germany), angle=(-180,+180,10))
```

```
In [23]: from shapely.affinity import scale

         def scaleme(g, xfactor, yfactor):
             frame = box(*g.bounds).exterior
             scaled = scale(g, xfact=xfactor, yfact=yfactor)
             return GeometryCollection((frame, scaled))

         interactive(scaleme, g=fixed(germany), xfactor=(0, 3, 0.05), yfactor=(0, 3, 0.05))
```

Um mehrere Geometrien gemeinsam darzustellen, ist GeometryCollection die einfachste Lösung.

```
In [24]: from shapely.geometry import GeometryCollection

In [25]: hamburg = Point(565811, 5933977).buffer(25000)

In [26]: GeometryCollection((germany, hamburg))

Out[26]:
```



1.4 Deutschland und Deutschland-Zwo

Die `relate`-Method einer Geometrie kann uns die DE-9IM im Bezug auf eine andere Geometrie zeigen:

```
In [27]: def relations(c1, c2, xoffset):
             translated = translate(c2, xoff=xoffset)

             print(c1.relate(translated))
             return GeometryCollection((
                 c1, translated
             ))
```

```
In [28]: interactive(relations, c1=fixed(germany), c2=fixed(germany), xoffset=(0,750000,75000))
```

Natürlich kann Shapely auch die üblichen geometrischen Operationen:

```
In [29]: def geometric_functions(c1, c2, xoffset, mode):
             c2_translated = translate(c2, xoff=xoffset)

             if mode == "intersection":
                 print("Intersects: {}".format(c1.intersects(c2_translated)))
                 result = c1.intersection(c2_translated)
             elif mode == "union":
                 result = c1.union(c2_translated)
             elif mode == "difference":
                 result = c1.difference(c2_translated)

             return GeometryCollection((
                 c1.exterior,
                 c2_translated.exterior,
                 result
             ))

         interactive(
             geometric_functions,
             c1=fixed(germany),
             c2=fixed(germany),
             xoffset=(0,750000,75000),
             mode=["intersection", "union", "difference"]
         )
```

1.5 Mit der Mittabstandstreue Azimutalprojektion herumspielen

```
In [30]: import fiona
         from shapely.geometry import shape, mapping, Point, GeometryCollection
         from shapely.ops import cascaded_union
         from fiona.crs import from_string
         from fiona.transform import transform_geom
```

```
from ipywidgets import interactive

with fiona.open("ne_110m_admin_0_countries.shp") as s:
    features = list(s)
    source_crs = s.crs

world_union = cascaded_union([shape(feature['geometry']) for feature in features if fea
world_union_simplified = world_union.buffer(1).buffer(-1)
world_union_simplified
```

Out[30]:

```
In [31]: def explore_aeqd(lat_0, lon_0, radius):
             target_crs = from_string("+proj=aeqd +R=6371000 +lat_0={} +lon_0={}".format(lat_0,
             buffer_around_center = Point(lon_0, lat_0).buffer(radius)
             world_cut = world_union_simplified.intersection(buffer_around_center)
             projected_geom = shape(transform_geom(source_crs, target_crs, mapping(world_cut)))
             projected_geom = projected_geom.buffer(0)  # buffer to fix self-intersections
             projected_buffer = shape(transform_geom(source_crs, target_crs, mapping(buffer_arou
             return GeometryCollection((projected_geom, projected_buffer.exterior))

In [32]: interactive(explore_aeqd, lat_0=(-45, 45), lon_0=(-180, 180, 2.5), radius=(0,180))
```

1.6 Zu jedem Baum das nächstgelegene Gebäude zeigen

```
In [33]: import rtree
         import fiona
         from shapely.geometry import *
         from ipywidgets import interactive, fixed

         with fiona.open("Strassenbaumkataster_HH_2017-01-06.shp") as trees_source:
             trees = list(trees_source)
         print(len(trees))
         trees_geometries = [shape(tree['geometry']) for tree in trees]
```

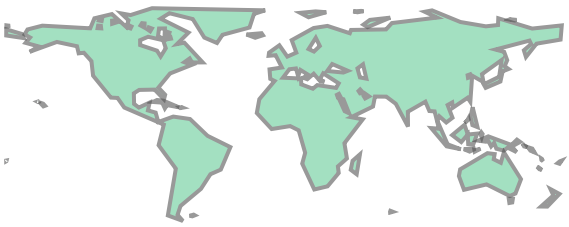
```
        with fiona.open("hamburg-latest-free_buildings_20170323_25832.shp") as buildings_source
            # this might be too big for memory! use buildings_source[::10] to sample only 10% o
            buildings = list(buildings_source)
        print(len(buildings))
        buildings_geometries = [shape(building['geometry']) for building in buildings]

        print("Builds index")
        r = rtree.index.Index()
        for i, building in enumerate(buildings_geometries):
            r.insert(i, building.bounds)
        print("Index built")
```

```
222667
299684
Builds index
Index built
```

```
In [34]: def treeexplorer(i):
             tree = trees_geometries[i]
             nearests = list(r.nearest(tree.bounds, 10))
             return GeometryCollection((
                 box(*GeometryCollection((MultiPolygon([buildings_geometries[i] for i in nearest
                 tree,
                 GeometryCollection([buildings_geometries[i] for i in nearests]),
                 LineString((tree.centroid, buildings_geometries[nearests[0]].centroid))
             ))

         interactive(treeexplorer, i=(0, len(trees_geometries)-1))
```

1.7 Geographic Slap

```
In [35]: import fiona
         from shapely.geometry import shape, mapping, Point, LineString, MultiLineString, Polygo
         from shapely.ops import cascaded_union
         from math import ceil
         from geographiclib.geodesic import Geodesic
         geod = Geodesic.WGS84

In [36]: # Damit es schneller rendert...
         world_union_simplified2 = world_union_simplified.simplify(2)
         world_union_simplified2

Out[36]:
```



In [37]:
```
def geographic_whip(slat, slon, length, direction):
    """
    slat, slon = source latlon
    direction = direction of "view" in degrees
    TODO schöner mit ArcDistance, das wäre metrisch ;)
    """

    #http://geographiclib.sourceforge.net/html/python/code.html#geographiclib.geodesic.
    l = geod.Line(slat, slon, direction)

    delta_arc = 0.5 # distance/delta arc degrees between points

    # get a arc distance that cleanly divides 360 degrees as i want to go once around
    n = int(ceil(length / delta_arc))
    delta_arc = length / n

    points = []

    for i in range(n+1):
        a = delta_arc * i
        g = l.ArcPosition(a, Geodesic.LATITUDE | Geodesic.LONGITUDE)
        points.append((g['lon2'], g['lat2']))

    return GeometryCollection((
        world_union_simplified2,
        Point(slon,slat),
        LineString(points)
    ))
```

In [38]:
```
interactive(geographic_whip, slat=(-90,90), slon=(-180,180), length=(0,360), direction=
```

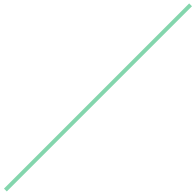
1.8 Wie spät ist es?

In [39]:
```
from shapely.geometry import LineString
```

In [40]:
```
p1 = Point(0, 0)
p2 = Point(1, 1)
l1 = LineString((p1, p2))
```

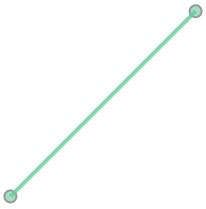
In [41]:
```
l1
```

Out[41]:



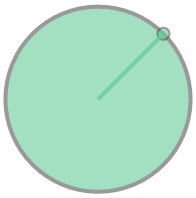
In [42]:
```
GeometryCollection((p1, p2, l1))
```

Out[42]:



In [43]:
```
GeometryCollection((p1.buffer(l1.length), p2, l1))
```

Out[43]:



In [44]:
```
from math import sin, cos, radians
```

In [45]:
```
def point_bydistanceandangle(p0, distance, angle):
    # angle in degrees please
    x = p0.x + distance * cos(radians(angle))
    y = p0.y + distance * sin(radians(angle))

    return Point(x, y)
```

```
In [46]: def clock(center, radius, angle):
             face = center.buffer(radius)
             endpoint = point_bydistanceandangle(center, radius, angle)
             arm = LineString((center, endpoint))
             return GeometryCollection((
                 face,
                 center,
                 arm,
                 endpoint
             ))

In [47]: interactive(clock, center=fixed(p1), radius=fixed(5), angle=(0,360))

In [48]: import time
         def clock():
             _, _, _, h, m, s, _, _, _ = time.localtime()
             print("{:02d}:{:02d}:{:02d}".format(h, m, s))

             #angle 90 is 12 o clock here
             angle_h = - h/12*360 + 90
             angle_m = - m/60*360 + 90
             angle_s = - s/60*360 + 90

             center = Point(0, 0)
             radius = 1
             face = center.buffer(radius)

             arm_h = LineString((center, point_bydistanceandangle(center, radius/2,   angle_h)))
             arm_m = LineString((center, point_bydistanceandangle(center, radius/1.5, angle_m)))
             arm_s = LineString((center, point_bydistanceandangle(center, radius/1,   angle_s)))
             return GeometryCollection((
                 face,
                 center,
                 arm_h,
                 arm_m,
                 arm_s
             ))

In [49]: from IPython import display
         from ipywidgets import Button

         button = Button(description='Klick mich!')

         def on_button_clicked(b):
             try:
                 while True:
                     # Danke an https://twitter.com/__phree__/status/844202341087764482 für den
                     display.clear_output(wait=True)
```

```
                display.display(clock())
                time.sleep(1)
            except KeyboardInterrupt:
                pass

        button.on_click(on_button_clicked)
        button
```

```
10:42:16
```