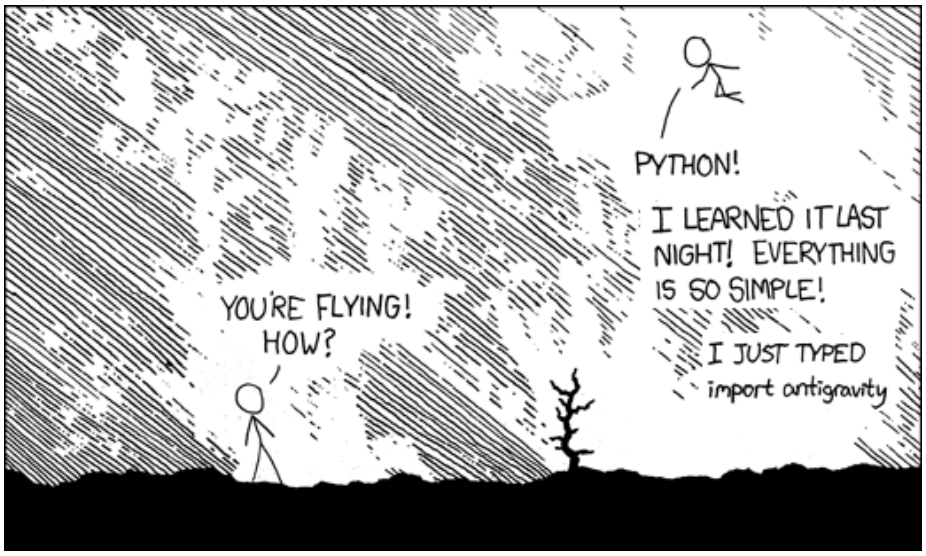
Geostat Summer School

GeoPython

Your Python Starter...

Python?

Its easy!



Its like R

Like this:

```
>>>  x = 1
>>>  y = 4
>>>  x * y
4
>>>  x + y
5
```

Not like this though:

```
>>>  x = [1, 2, 3]
>>>  y = 3 + x
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'list'
```

Like this:

```
>>>  x = [1, 2, 3]
>>>  y = [3 + z for z in x]
>>>  y
[4, 5, 6]
```

Functions

def

- Define functions:

```
>>>  def poly(x, a, b, c):
...    y = [a * z**2 + b * z + c for z in x]
...    return y
...
```

- Call functions:

```
>>>  poly([1,2,3], 2,-1,3)
[4, 9, 18]
>>>
>>>  poly([1,2,3], a=2, b=-1, c=3)
[4, 9, 18]
>>>
>>>  poly([1,2,3], c=2, b=-1, 3)
  File "<console>", line 1
SyntaxError: non-keyword arg after keyword arg
>>>
>>>  poly([1,2,3], 2, b=-1, c=3)
[4, 9, 18]
```

Args

Defaults etc...

```
>>>  def foo(myparameter=99):
...     return myparameter*2
...
>>>  foo()
198
>>>  foo(100)
200
>>>  foo(myp=100)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: foo() got an unexpected keyword argument 'myp'
>>>  foo(myparameter=100)
200
>>>  foo(myparameterandabit=100)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: foo() got an unexpected keyword argument 'myparameterandabit'
```

Loops

`for` loops

```
>>>  word = ["B", "A", "R", "R", "Y"]
>>>  for letter in word:
...     print letter
...
B
A
R
R
Y
```

Nested

Loops in loops

```
>>>  n = len(word)
>>>  for ia in range(0,n):
```

```
...     a = word[ia]
...     for ib in range(ia+1, n):
...       b = word[ib]
...       print a,b
...
B A
B R
B R
B Y
A R
A R
A Y
R R
R Y
R Y
```

That's a fairly common pattern, we've cluttered it up. All we really want to do is:

```
for (a,b) in pairs(word):
  print a,b
```

Generators

Advanced stuff made easy

```
>>>  def pairs(v):
...       n = len(v)
...       for i in range(0,n):
...           a = v[i]
...           for j in range(i+1, n):
...               b=v[j]
...               yield (a,b)
...
>>>  for (a,b) in pairs(word):
...     print a,b
...
B A
B R
B R
B Y
A R
A R
```

```
A Y
R R
R Y
R Y
```

imports

get code from files

```
#
# this is samples.py in my working directory
#
def bar(x):
  return x*2

def baz(x):
  return x*3
```

```
>>>  import samples
>>>  samples.bar(99)
198
>>>
>>>  from samples import baz
>>>  baz(100)
300
>>>
>>>  # if I edit samples.py....
>>>
>>>  reload(samples)
<module 'samples' from '/home/rowlings/Work/Teaching/GeostatSummerSchool/2014/Site/geostat/s
```

Library Modules

Get installed module code

```
>>>  import os
>>>  os.path.realpath(".")
'/home/rowlings/Work/Teaching/GeostatSummerSchool/2014/Site/geostat'
>>>  os.path.join("foo","bar","baz")
'foo/bar/baz'
>>>  os.uname()
('Linux', 'barry-OptiPlex-755', '3.8.0-31-generic', '#46-Ubuntu SMP Tue Sep 10 20:03:44 UTC
```

Python comes with a complete standard library. R didn't have a full file name handling package until 2014 (the `pathological` package on github).

Help!

How to get help

```
>>>  help(os.uname)
Help on built-in function uname in module posix:

uname(...)
    uname() -> (sysname, nodename, release, version, machine)

    Return a tuple identifying the current operating system.
```

Unlike R, you can get help from objects!

Methods

What can I do with X?

```
>>>  x="hello world"
>>>  dir(x)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__',
>>>  x.title()
'Hello World'
>>>  x.upper()
'HELLO WORLD'
>>>  x.split()
['hello', 'world']
>>>  help(x.split)
Help on built-in function split:

split(...)
    S.split([sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are removed
    from the result.
```

Python Objects

Lots of things are objects

- `x = "hello world"` creates an object `x`
- `x.split()` runs the `split` "method" on `x`, returns a list. `x` doesn't change...
- but methods *can* change the object they run on.

```
>>> z = [9,6,2,3]
>>> z.sort()
>>> z
[2, 3, 6, 9]
```

- They can return a value *and* change the object:

```
>>> last = z.pop()
>>> last
9
>>> z
[2, 3, 6]
```

A sample class

Point object

```
>>> class Point(object):
...   def __init__(self,x,y):
...     self.x = x
...     self.y = y
...   def coord(self):
...     return [self.x, self.y]
...   def shift(self, dx, dy):
...     self.x = self.x + dx
...     self.y = self.y + dy
...
>>> p = Point(2.4, 4.5)
>>>
>>> p.x
2.4
>>>
>>> p.coord()
```

```
[2.4, 4.5]
>>>
>>>  p.shift(1000,2000)
>>>  p.coord()
[1002.4, 2004.5]
>>>
```

Distance

Pythagoras

```
>>>  import math
>>>  def pythagoras(p0, p1):
...     return math.sqrt((p1.x-p0.x)**2 + (p1.y-p0.y)**2)
...
>>>  pythagoras(Point(0,0),Point(3,4))
5.0
```

Sphere Distance

Spherical Distance

```
>>>  # From: https://gist.github.com/gabesmed/1826175
>>>
>>>  EARTH_RADIUS = 6378137 # radius in metres
>>>
>>>  points = [
...             Point(40.750307,-73.994819),
...             Point(40.749641,-73.99527)
...           ]
>>>
>>>  def great_circle_distance(latlong_a, latlong_b):
...      lat1, lon1 = latlong_a.coord()
...      lat2, lon2 = latlong_b.coord()
...
...      dLat = math.radians(lat2 - lat1)
...      dLon = math.radians(lon2 - lon1)
...      a = (math.sin(dLat / 2) * math.sin(dLat / 2) +
...              math.cos(math.radians(lat1)) * math.cos(math.radians(lat2)) *
...              math.sin(dLon / 2) * math.sin(dLon / 2))
...      c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
...      d = EARTH_RADIUS * c
```

```
...
...      return d
...
>>>  great_circle_distance(points[0], points[1])
83.32536285505579
```

Ellipses?

Vincenty's Formula (wikipedia)

Part one...

$$\sin\sigma = \sqrt{(\cos U_2 \sin\lambda)^2 + (\cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos\lambda)^2}$$

$$\cos\sigma = \sin U_1 \sin U_2 + \cos U_1 \cos U_2 \cos\lambda$$

$$\sigma = \arctan \frac{\sin\sigma}{\cos\sigma}$$ [1][2]

$$\sin\alpha = \frac{\cos U_1 \cos U_2 \sin\lambda}{\sin\sigma}$$ [3]

$$\cos^2\alpha = 1 - \sin^2\alpha$$

$$\cos(2\sigma_m) = \cos\sigma - \frac{2\sin U_1 \sin U_2}{\cos^2\alpha}$$ [4]

$$C = \frac{f}{16}\cos^2\alpha[4 + f(4 - 3\cos^2\alpha)]$$

$$\lambda = L + (1-C) f \sin\alpha \left\{\sigma + C\sin\sigma\left[\cos(2\sigma_m) + C\cos\sigma(-1 + 2\cos^2(2\sigma_m))\right]\right\}$$

Serious numbers

Use numpy for maths

```
>>>  import numpy as np
>>>  [math.sin(x) for x in [1,2,3]]
[0.8414709848078965, 0.9092974268256817, 0.1411200080598672]
>>>  np.sin([1,2,3])
array([ 0.84147098,  0.90929743,  0.14112001])
>>>  np.random.poisson(10,20)
array([11, 12, 10,  8, 13, 11, 14, 10, 10, 13,  7, 15,  8, 12, 10, 10,  8,
        6, 11, 11])

>>>  m = np.matrix([[1,2], [3,4]])
>>>  m
```

```
matrix([[1, 2],
        [3, 4]])
>>>  m * m
matrix([[ 7, 10],
        [15, 22]])
>>>  m * m.transpose()
matrix([[ 5, 11],
        [11, 25]])
>>>  m * m.I
matrix([[  1.00000000e+00,   0.00000000e+00],
        [  8.88178420e-16,   1.00000000e+00]])
```

External Libraries

Geopy

```
>>>  import geopy
>>>  from geopy.distance import great_circle
>>>
>>>  d = great_circle(points[0].coord(), points[1].coord())
>>>  d
Distance(0.0832555738102)
>>>  help(d)
Help on great_circle in module geopy.distance object:

class great_circle(Distance)
 |  Use spherical geometry to calculate the surface distance between two
 |  geodesic points. This formula can be written many different ways,
 |  including just the use of the spherical law of cosines or the haversine
 |  formula.
 |
 |  Set which radius of the earth to use by specifying a 'radius' keyword
 |  argument. It must be in kilometers. The default is to use the module
 |  constant `EARTH_RADIUS`, which uses the average great-circle radius.
 |
 |  Example::
 |
 |      >>> from geopy.distance import great_circle
 |      >>> newport_ri = (41.49008, -71.312796)
 |      >>> cleveland_oh = (41.499498, -81.695391)
 |      >>> great_circle(newport_ri, cleveland_oh).miles
 |      537.1485284062816
 |
 |  Method resolution order:
```

```
|      great_circle
|      Distance
|      __builtin__.object
|
|  Methods defined here:
|
|  __init__(self, *args, **kwargs)
|
|  destination(self, point, bearing, distance=None)
|      TODO docs.
|
|  measure(self, a, b)
|
|  ----------------------------------------------------------------------
|  Methods inherited from Distance:
|
|  __abs__(self)
|
|  __add__(self, other)
|
|  __bool__ = __nonzero__(self)
|
|  __cmp__(self, other)
|
|  __div__(self, other)
|
|  __mul__(self, other)
|
|  __neg__(self)
|
|  __nonzero__(self)
|
|  __repr__(self)
|
|  __str__(self)
|
|  __sub__(self, other)
|
|  __truediv__ = __div__(self, other)
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from Distance:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
```

```
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  feet
 |
 |  ft
 |
 |  kilometers
 |
 |  km
 |
 |  m
 |
 |  meters
 |
 |  mi
 |
 |  miles
 |
 |  nautical
 |
 |  nm

>>>
>>>  d.meters
83.255573810242
>>>
>>>  d.feet
273.1482079345222
>>>
>>>  geopy.distance.EARTH_RADIUS = EARTH_RADIUS/1000
>>>  great_circle(points[0].coord(), points[1].coord()).meters
83.32357305730429
```

geopy again

Vincenty distance

```
>>>  from geopy.distance import vincenty
>>>  great_circle(points[0].coord(), points[1].coord()).m
83.32357305730429
>>>  vincenty(points[0].coord(), points[1].coord()).m
83.19003856394464
```

More geopy magic

geocoding

```
>>>  from geopy.geocoders import osm
>>>  g = osm.Nominatim(timeout=10)
>>>  home = g.geocode("Lancaster, UK")
>>>  away = g.geocode("Bergen")
>>>  vincenty(home.point, away.point).miles
532.7862059372253
>>>  vincenty(home.point, away.point).km
857.4362841353374
```

fiona

Read data with fiona

```
>>>  import fiona
>>>  england = fiona.open("./data/England/engos.shp","r")
>>>  england.meta
{'crs': {u'lon_0': -2, u'k': 0.9996012717, u'datum': u'OSGB36', u'y_0': -100000, u'no_defs':
>>>  england.meta['crs']
{u'lon_0': -2, u'k': 0.9996012717, u'datum': u'OSGB36', u'y_0': -100000, u'no_defs': True, u
>>>  england.meta['schema']['properties']
OrderedDict([(u'ADMIN_NAME', 'str:16'), (u'AREA', 'float:12.3'), (u'PERIMETER', 'float:12.3'
```

features

get the features

```
>>>  features = list(england)
>>>  features[31]['properties']
OrderedDict([(u'ADMIN_NAME', u'Greater London'), (u'AREA', 0.208), (u'PERIMETER', 2.344), (u
>>>  features[32]['geometry'].keys()
['type', 'coordinates']
>>>  features[32]['geometry']['type']
'Polygon'
>>>  features[32]['geometry']['coordinates'][0][:10]
[(331218.2373748748, 158030.96091701294), (331552.45641719806, 159850.56801485896), (331184.
```

processing with shapely

example

```
>>>  from shapely.geometry import mapping, shape
>>>
>>>  import fiona
>>>
>>>  input_shp = "./data/England/engos.shp"
>>>  output_shp = "./data/England/buffered.shp"
>>>  width = 10000
>>>
>>>  def bufferinout(input_shp, output_shp, width):
...       input = fiona.open(input_shp, "r")
...       schema = { 'geometry': 'Polygon', 'properties': { 'name': 'str' } }
...       output =  fiona.open(output_shp, "w", "ESRI Shapefile", schema, crs = input.crs)
...       for feature in input:
...            output.write({
...            'properties': {
...                'name': feature['properties']['ADMIN_NAME']
...            },
...            'geometry': mapping(shape(feature['geometry']).buffer(width))
...            })
...
>>>  bufferinout(input_shp, output_shp, width)
```

Scripting

Command-line scripts

With this in `buffering.py` for example:

```
import sys

import fiona

def bufferinout(input, output, width):
  " compute the buffer... "
  ....

if __name__=="__main__":
  input = sys.argv[1]
```

```
  output = sys.argv[2]
  width = float(sys.argv[3])
  bufferinout(input, output, width)
```

Can then:

- `python buffering.py england.shp buffer10k.shp 10000` on the command line
- `from buffering import bufferinout` in python and use that function.

Shell power

back to bash...

```
for width in 10000 20000 30000 40000 ; do
 python buffering.py england.shp buffer${width}.shp $width
done
```

Or in Python:

```
from buffering import bufferinout
for width in [10000, 20000, 30000, 40000]:
     bufferinout("england.shp","buffer%s.shp" % width, width)
```

Geopandas

The future

Geopandas

Like Spatial Data Frames

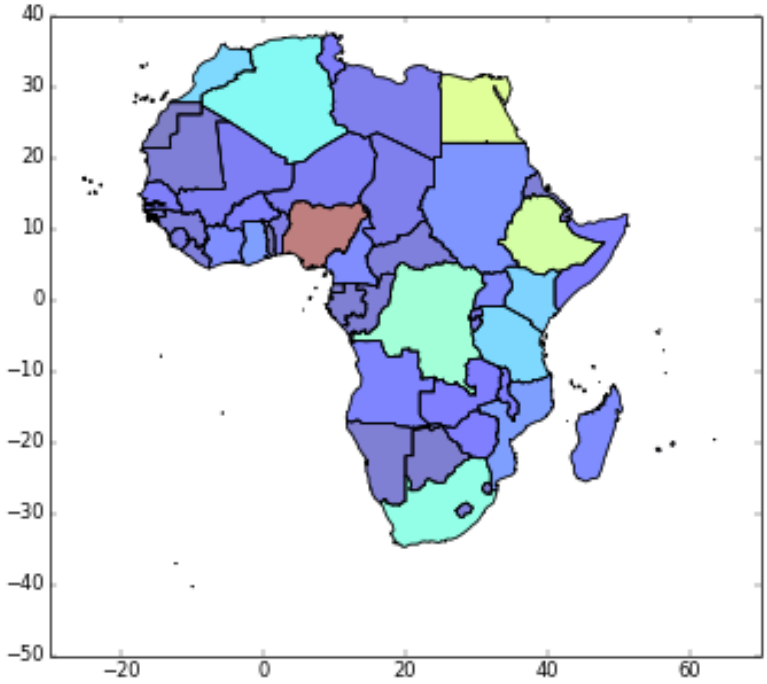
```
>>>  import geopandas as gpd
>>>  africa = gpd.read_file("./data/Africa/africa.shp")
>>>  africa[:5]
     CNTRY_NAME  COUNT FIRST_CONT FIRST_FIPS       FIRST_REGI  SUM_POP_AD  \
0       Algeria     48     Africa          AG  Northern Africa    34222570
1        Angola     18     Africa          AO    Middle Africa    11527258
```

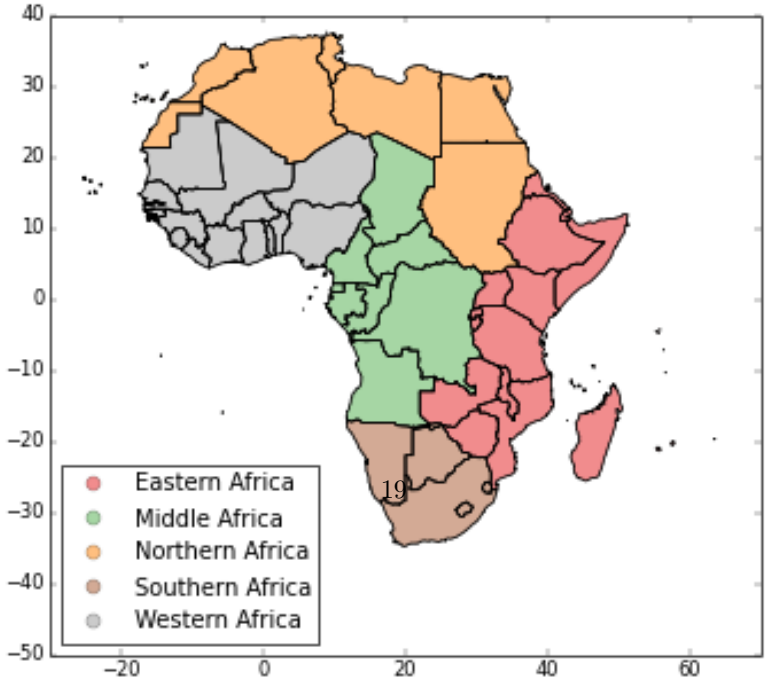
```
2         Benin      6     Africa         BN   Western Africa     5175394
3      Botswana     10     Africa         BC  Southern Africa     1185250
4  Burkina Faso     30     Africa         UV   Western Africa    10817069

    SUM_SQKM_A  SUM_SQMI_A                                          geometry
0  2320972.339  896127.445  POLYGON ((2.963609933853149 36.80221557617188,...
1  1252420.770  483559.643  (POLYGON ((11.77499961853027 -16.8047256469726...
2   116514.769   44986.354  POLYGON ((2.484417915344238 6.340485572814941,...
3   580011.123  223942.302  POLYGON ((26.16782760620117 -24.66396713256836...
4   273719.207  105682.987  POLYGON ((-1.003023386001587 14.8400993347168,...

[5 rows x 9 columns]
```

plotting

plot method

```
>>>  import matplotlib.pyplot as plt
>>>  fig = africa.plot(column="SUM_POP_AD")
>>>  plt.show()
>>>  plt.cla()
>>>  fig = africa.plot(column="FIRST_REGI",legend=True)
>>>  plt.show()
```

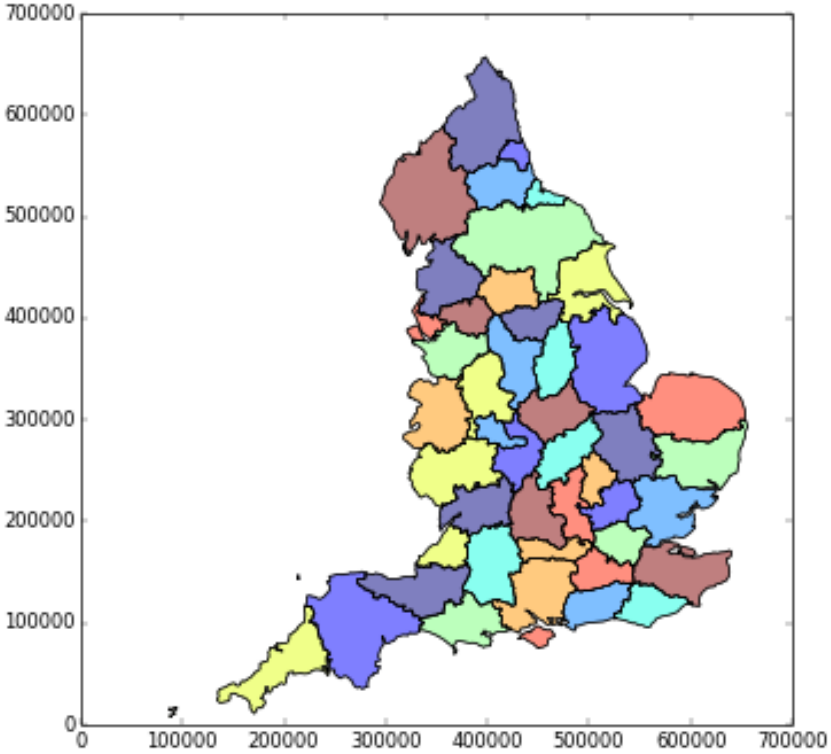

40
30
20
10
0
−10
−20
−30
−40
−50
−20
0
20
40
60

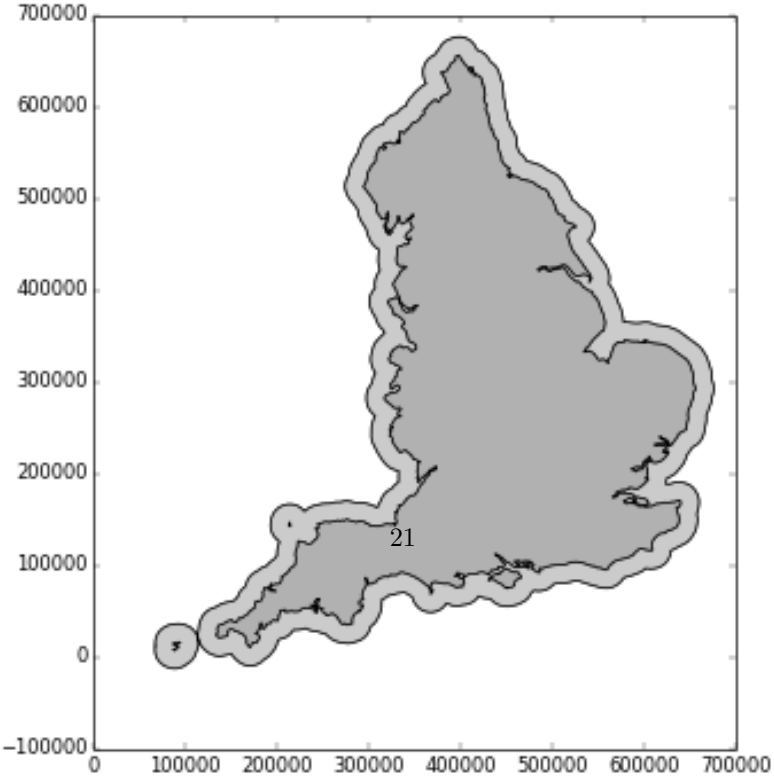

40
30
20
10
0
−10
−20
−30
−40
−50
−20
0
20
40
60
Eastern Africa
Middle Africa
Northern Africa
Southern Africa
Western Africa

Buffering Geopandas

GeoPandas methods for geometry

```
>>>  from geopandas import GeoSeries
>>>  england = gpd.read_file("./data/England/engos.shp")
>>>  fig = england.plot()
>>>  plt.show()
>>>  plt.cla()
>>>  coast = GeoSeries(england.geometry.unary_union)
>>>  coastal_buffer = GeoSeries(coast.buffer(20000))
>>>  fig = coastal_buffer.plot()
>>>  fig = coast.plot()
>>>  plt.show()
```


700000
600000
500000
400000
300000
200000
100000
0
0
100000
200000
300000
400000
500000
600000
700000


700000
600000
500000
400000
300000
200000
100000
0
−100000
0
100000
200000
300000
400000
500000
600000
700000

JS Mapping

Packages

- mplleaflet

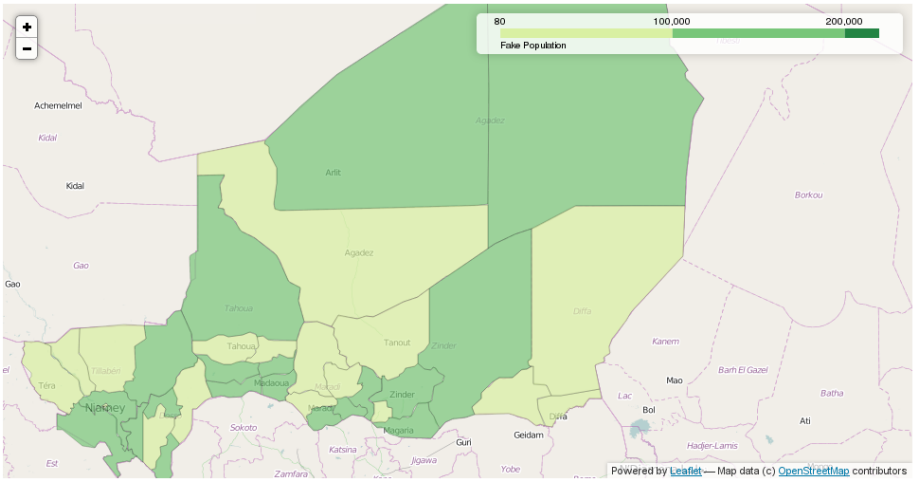
```
import mplleaflet
niger = gpd.read_file("./data/Africa/niger.shp")
niger.plot()
# write an HTML file and show in browser
mplleaflet.show()
```

- folium

```
# convert to geoJSON file
open("niger.json","w").write(niger.to_json())

map_osm = folium.Map(location=[17,10])

# add map to output
map_osm.geo_json("./niger.json", data=niger,
         columns=['ADM2','POP'], key_on='feature.properties.ADM2',
         fill_color='YlGn', fill_opacity=0.7, line_opacity=0.2,
         legend_name="Fake Population")

# create and view in browser
map_osm.create_map("niger.html")
```

Leaflet map

From folium

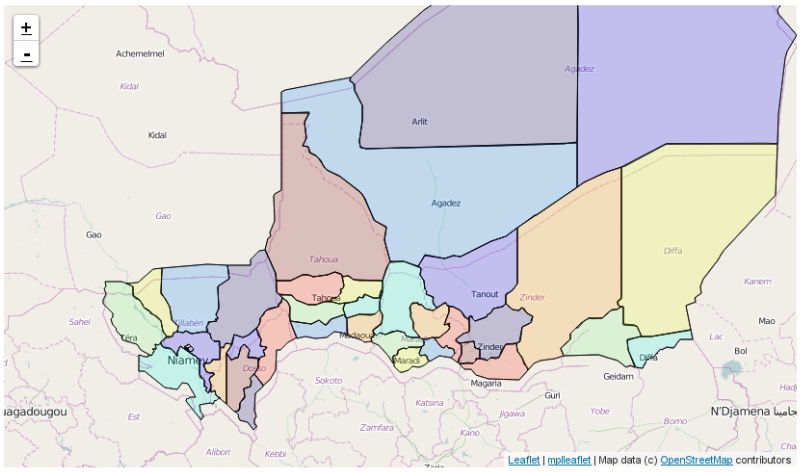


ipython

notebooks

In [5]:
```
import matplotlib.pyplot as plt
import mplleaflet
import geopandas as gpd

africa = gpd.read_file("./data/Africa/Niger/newNiger.shp")
ax = africa.plot()

mplleaflet.display()
```

Out[5]:



pysal

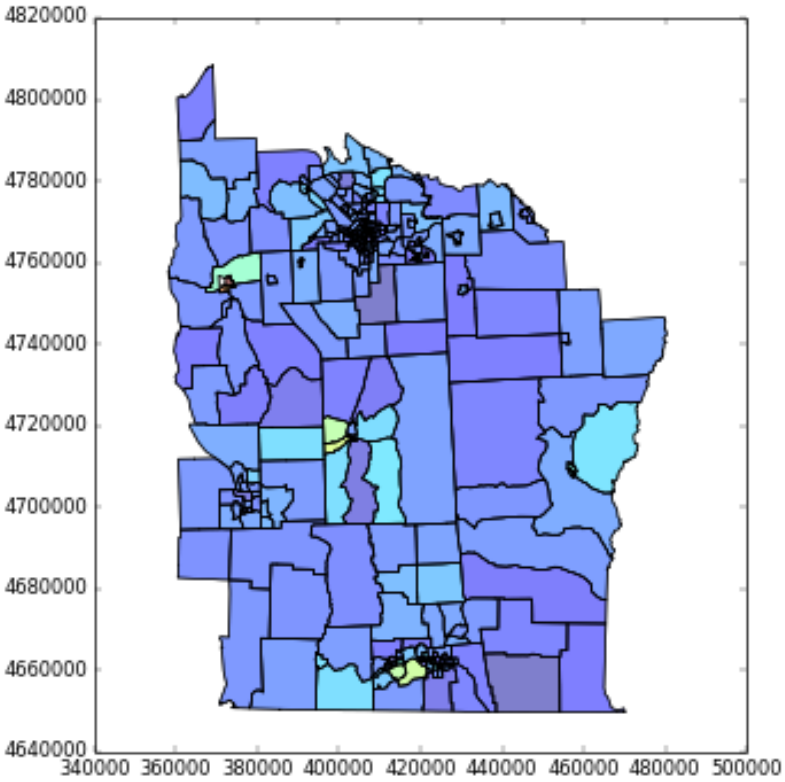
Local Moran calculation

```
>>>  import geopandas as gpd
>>>  import numpy as np
>>>  import pysal
>>>  shapefile = "data/NewYork/NY8_utm18.shp"
>>>  galfile = "data/NewYork/NY_nb.gal"
>>>  spdf = gpd.read_file(shapefile)
>>>
>>>  y = np.array(spdf['Cases'])
>>>  w = pysal.open(galfile).read()
>>>  lm = pysal.Moran_Local(y,w,transformation="V")
>>>  lm.Is[:5]
array([ 0.56778845,  0.70374966, -0.51771761, -0.19487455, -0.17114584])
```

Map

Plot the I values

```
>>>  spdf['I']=lm.Is
>>>  spdf.plot(column="I")
<matplotlib.axes.AxesSubplot object at 0x60dbc10>
>>>  plt.show()
```



pysal

Features

- esda (moran, geary. . .)

- smoothing (empirical bayes...)
- regression with spatial weights (ols, probit...)

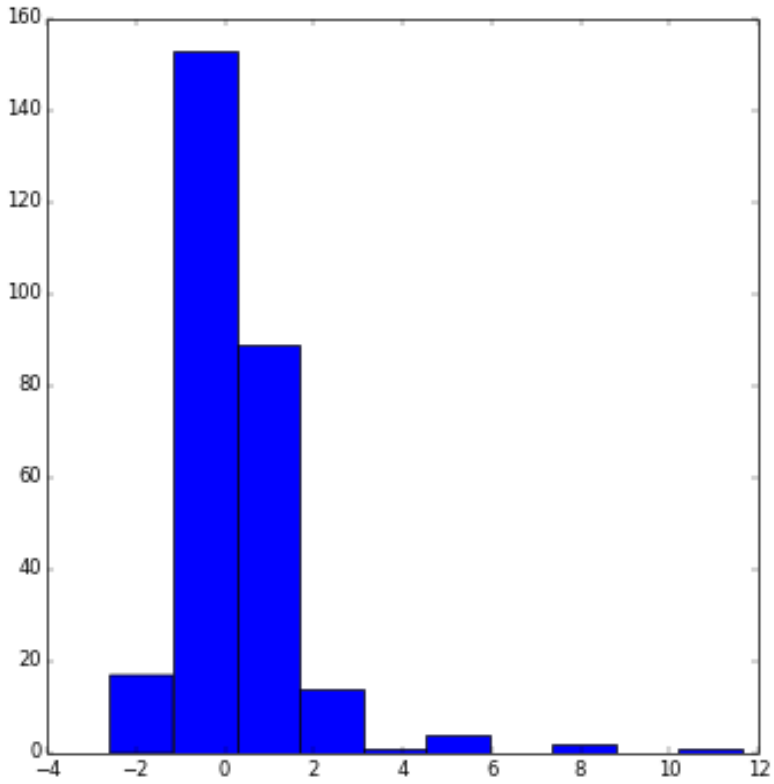
Unfeatures

- graphics
- summaries
- documentation?

Rpy2

Fill in the gaps

```
>>>  import rpy2.robjects as robjects
>>>  from rpy2.robjects.packages import importr
>>>  spdep = importr("spdep")
>>>  NY_nb = spdep.read_gal(galfile, region=range(281))
>>>  lmR = spdep.localmoran(robjects.FloatVector(spdf['Cases']), listw = spdep.nb2listw(NY_r
>>>  fig = plt.hist(lmR.rx(True,"Z.Ii"))
>>>  plt.show()
```



Summary

Stuff you might want to look into...

- Python
- ipython notebooks are cool!
- Numpy for numeric matrix/raster ops
- Scipy - scientific python
- shapely, fiona, rasterio for data and geometry
- Geopandas for spatial data
- Pysal for spatial statistics
- pymc for MCMC calculations (like BUGS)
- *LOTS* of machine learning stuff
- Rpy2 when all else fails