Geospatial data processing with Python

March. 22, 2019

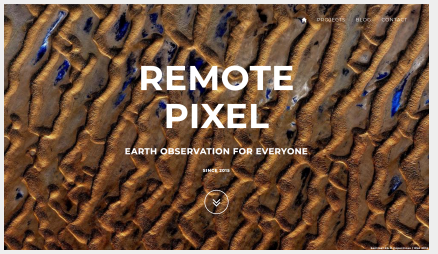
development SEED

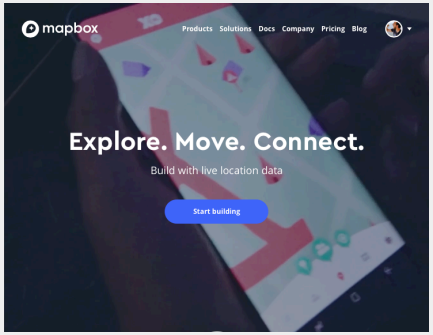
Vincent Sarago

vincentsarago

@_VincentS_

- Former geologist
- Geospatial developer
- Self taught Python dev
- Full stack





developmentSEED

Why Python ?

development SEED

Community

- Everyone use it
- Tons of Open source modules

developmentSEED

A scientific language

- Numpy
- Jupyter
- TensorFlow
- Spark
- Pandas
- Scipy
- …

developmentSEED

A Web language

- Django
- Flask
- FastAPI
- …

development SEED

Geospatial processing

- Read/Write files in various format
- Perform spatial prediction (e.g. Point in Polygon), operation (e.g buffer), computation (e.g length)
- Projection transformation (e.g from WGS84 to UTM)

Geospatial Libs

Capability	C/C++
Spatial Operations	GEOS
Vector I/O	GDAL (OGR)
Raster Operations and I/O	GDAL
Projections	PROJ.4
Spatial Indexing	libspatialindex

ref: http://jwass.github.io/maptime-boston-python/slides/

Vector

development SEED

GDAL (I/O)

pip install GDAL

« GDAL is a translator library for raster and vector geospatial data formats. »

https://www.gdal.org

- Not user friendly
- Lack of documentation
- Low level access to GDAL api

```
from osgeo import ogr, osr
file = "my-shape.shp"
```

```
if file.endswith('.shp'):
    driver = ogr.GetDriverByName('ESRI Shapefile')
elif file.endswith('.geojson'):
    driver = ogr.GetDriverByName('GeoJSON')

inShp = driver.Open(file, 0)
```

```
inShp.GetLayerCount()
```

```
1
```

```
print(inLyr.GetSpatialRef())
```

```
PROJCS["NAD83 / Statistics Canada Lambert",
    GEOGCS["NAD83",
        DATUM["North_American_Datum_1983",
            SPHEROID["GRS 1980",6378137,298.257222101,
                AUTHORITY["EPSG","7019"]],
            TOWGS84[0,0,0,0,0,0,0],
            AUTHORITY["EPSG","6269"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4269"]],
    PROJECTION["Lambert_Conformal_Conic_2SP"],
    PARAMETER["standard_parallel_1",49],
    PARAMETER["standard_parallel_2",77],
    PARAMETER["latitude_of_origin",63.390675],
    PARAMETER["central_meridian",-91.86666666666666],
    PARAMETER["false_easting",6200000],
    PARAMETER["false_northing",3000000],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["Easting",EAST],
    AXIS["Northing",NORTH],
    AUTHORITY["EPSG","3347"]]
```

```
for feature in inShp.GetLayer():
    print(feature.GetGeometryRef())
```

```
POLYGON ((5700075.94857 1682980.84857,5700146.794285 1682917.9,5700203.46857 1682968.474285,5700242.657145 1682997.03
143,5700265.922855 1683008.39143,5700317.374285 1683021.585715,5700392.49143 1683023.102855,5700687.297145 1682989.05
4285,5700618.325715 1682330.705715,5700606.5 1682228.254285,5700555.434285 1681755.814285,5700370.554285 1680344.5628
55,5700540.06 1680324.914285,5702099.11143 1680144.18,5702238.125715 1680128.737145,5702515.84857 1679713.82857,57025
66.19143 1679646.19143,5702620.065715 1679592.15143,5702674.285715 1679550.074285,5702763.802855 1679493.00857,570310
7.06857 1679275.454285,5703200.64 1679218.94,5703329.03143 1679125.52857,5703384.77143 1679070.277145,5703424.72 1679
016.805715,5703482.157145 1678923.385715,5703505.657145 1678874.77143,5703590.38857 1678727.79143,5703924.225715 1678
154.374285,5704199.857145 1677673.245715,5704581.425715 1677007.777145,5704677.697145 1676839.877145,5704732.26857 16
76729.794285,5704762.47143 1676607.457145,5704764.58 1676475.297145,5704684.374285 1675756.325715,5704631.17143 16752
91.125715,5704609.805715 1675109.434285,5704596.12 1674979.042855,5704588.58 1674849.95143,5704606.942855 1674728.948
57,5704647.81143 1674621.39143,5704708.42857 1674520.617145,5704784.437145 1674441.08857,5704857.077145 1674384.93714
5,5704929.634285 1674345.777145,5704958.24 1674333.562855,5705028.84857 1674312.62,5705125.02 1674296.8,5705774.21143
1674223.72857,5706131.78 1674183.48,5707328.817145 1674046.79143,5707430.914285 1674030.317145,5707523.114285 1673996
.954285,5707603.125715 1673952.965715,5707680.14857 1673891.32857,5707728.67143 1673842.89143,5707650.84 1673150.9571
45,5707564.005715 1672370.097145,5708340.942855 1672283.714285,5708166.237145 1670668.02857,5707993.16857 1669049.091
43,5707821.08 1667429.954285,5707752.31143 1666826.78,5707603.38857 1665458.665715,5707493.965715 1664423.462855,5707
504.005715 1664299.357145,5707537.317145 1664186.634285,5707545.74857 1664172.685715,5707592.7 1664083.422855,5707660
.397145 1664000.82,5707713.602855 1663951.717145,5707844.84 1663836.11143,5708175.265715 1663546.997145,5708264.02285
5 1663456.034285,5708338.725715 1663355.642855,5708398.26857 1663244.965715,5708739.725715 1662395.702855,5709045.334
```

```
inShp = None
```

developmentSEED

Fiona (I/O)

pip install fiona

« To eliminate unnecessary complication. Fiona aims to be simple to understand and use, with no gotchas. »

https://fiona.readthedocs.io/en/latest/manual.html

- Better python code
- Better documentation
- Well maintained
- **Shipped with GDAL** (wheels)
- Plugins

https://github.com/Toblerity/Fiona

```
import fiona
file = "my-shape.shp"
```

```
with fiona.open(file) as src:
    print(src.meta)
    print(len(src))
    print(src[1])
```

```
{'driver': 'ESRI Shapefile', 'schema': {'properties': OrderedDict([('DAUID', 'str:8'), ('PRUID', 'str:2'), ('PRNAME'
, 'str:100')]), 'geometry': 'Polygon'}, 'crs': {'init': 'epsg:3347'}, 'crs_wkt': 'PROJCS["NAD83 / Statistics Canada
Lambert",GEOGCS["NAD83",DATUM["North_American_Datum_1983",SPHEROID["GRS 1980",6378137,298.257222101,AUTHORITY["EPSG"
,"7019"]],TOWGS84[0,0,0,0,0,0,0],AUTHORITY["EPSG","6269"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degr
ee",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4269"]],PROJECTION["Lambert_Conformal_Conic_2SP"]
,PARAMETER["standard_parallel_1",49],PARAMETER["standard_parallel_2",77],PARAMETER["latitude_of_origin",63.390675],P
ARAMETER["central_meridian",-91.86666666666666],PARAMETER["false_easting",6200000],PARAMETER["false_northing",300000
0],UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northing",NORTH],AUTHORITY["EPSG","3347"]]'}
4657
{'type': 'Feature', 'id': '1', 'properties': OrderedDict([('DAUID', '46170091'), ('PRUID', '46'), ('PRNAME', 'Manito
ba')]), 'geometry': {'type': 'Polygon', 'coordinates': [[(5701064.9, 1682942.305715), (5701594.56857, 1682880.602855
), (5701609.925714999, 1682883.862855), (5701761.771430001, 1682865.757145), (5701834.9428550005, 1682853.92857), (5
701863.84, 1682849.254285), (5701902.542855, 1682846.885715), (5702010.702855, 1682834.43143), (5702080.522855001, 1
682826.39143), (5702168.742854999, 1682816.234285), (5702307.985715, 1682800.2), (5702345.817145, 1682798.93143), (5
702433.774285, 1682788.1), (5702493.422855, 1682780.757145), (5702537.411429999, 1682775.34), (5702585.234285, 16827
69.45143), (5702632.634285, 1682763.614285), (5702673.42, 1682762.017145), (5702714.862855, 1682757.354285), (570275
6.854285, 1682748.62), (5702863.625715001, 1682738.114285), (5702967.365715, 1682727.905715), (5703055.837145001, 16
82714.934285), (5703092.04857, 1682713.397145), (5703098.602855001, 1682713.117145), (5703281.462855, 1682692.517145
), (5703533.194285, 1682664.162855), (5703636.874284999, 1682652.20857), (5703865.205715001, 1682625.762855), (57039
34.12857, 1682616.46), (5704044.34857, 1682601.58), (5704338.4028549995, 1682568.465715), (5704484.705715001, 168255
2.01143), (5704755.808569999, 1682521.457145), (5704789.68857, 1682520.642855), (5704934.274285, 1682501.12), (57050
03.374284999, 1682492.58), (5705038.245715, 1682491.657145), (5705136.922855, 1682480.54857), (5705169.154285001, 16
82473.914285), (5705196.014285, 1682472.894285), (5705253.02857, 1682464.47143), (5705621.965715, 1682421.93143), (5
705704.09143, 1682414.685715), (5705973.36857, 1682383.37143), (5706036.077145, 1682372.32), (5706073.562855, 168235
9.11143), (5706093.177145001, 1682346.90857), (5706110.477144999, 1682331.965715), (5706126.34857, 1682313.194285),
(5706140.025715, 1682292.66857), (5706172.831429999, 1682229.017145), (5706187.797145, 1682184.36857), (5706123.3257
15, 1682117.674285), (5706095.537145, 1682083.834285), (5706086.29143, 1682063.882855), (5706079.905715, 1682042.62)
, (5706058.3885699995, 1681949.114285), (5706050.42857, 1681905.037145), (5706051.93143, 1681882.88857), (5706058.36
5715, 1681860.174285), (5706068.97143, 1681838.994285), (5706103.345714999, 1681798.157145), (5706066.02, 1681768.37
4285), (5706024.537145, 1681746.07143), (5705975.602855001, 1681728.597145), (5705864.145715, 1681697.16857), (57058
39.779999999, 1681684.925715), (5705816.07143, 1681669.60857), (5705777.31143, 1681636.0), (5705732.16, 1681581.12),
(5705645.002855, 1681446.037145), (5705534.777145, 1681283.562855), (5705517.63143, 1681255.52), (5705475.66, 168120
2.282855), (5705434.15143, 1681161.985715), (5705408.134285, 1681143.92857), (5705322.411429999, 1681092.617145), (5
705190.585715, 1681018.762855), (5704784.0457150005, 1680791.0), (5704652.585715, 1680717.34857), (5704534.14857, 16
80651.665715), (5704478.842855, 1680620.994285), (5704424.525715, 1680600.125715), (5704394.794284999, 1680593.48285
5), (5704365.394285, 1680589.79143), (5704302.67143, 1680591.862855), (5704270.5571449995, 1680599.48), (5704238.88,
1680611.037145), (5704207.534285, 1680625.557145), (5704183.205715001, 1680640.242855), (5704149.137145, 1680666.117
145), (5704123.84, 1680689.94857), (5704061.102855001, 1680760.65143), (5704052.9, 1680769.897145), (5703972.62857,
1680846.885715), (5703910.957145, 1680893.80857), (5703844.78857, 1680936.237145), (5703810.05143, 1680956.134285),
(5703712.042855, 1681004.72), (5703703.1, 1681009.154285), (5703662.0, 1681027.997145), (5703620.4942850005, 1681047
.025715), (5703561.5085700005, 1681074.07143), (5703526.87143, 1681085.96857), (5703454.74, 1681102.082855), (570338
0.282855, 1681106.482855), (5703304.497145, 1681099.02857), (5702605.222855, 1680994.925715), (5702527.674285, 16809
80.682855), (5702452.9542849995, 1680956.12), (5702418.814285, 1680936.985715), (5702387.42, 1680915.54), (5702358.6
5143, 1680890.797145), (5702332.5057149995, 1680862.76857), (5702309.20857, 1680833.405715), (5702204.037145, 168068
9.36), (5702133.691430001, 1680588.357145), (5702099.085715, 1680547.294285), (5702057.359999999, 1680514.014285), (
5702004.337145001, 1680478.01143), (5701936.21143, 1680582.817145), (5701691.1, 1680947.685715), (5701503.12857, 168
1227.04857), (5701442.5085700005, 1681309.83143), (5701309.422855, 1681476.73143), (5701129.237145, 1681699.402855),
(5701001.542855, 1681857.205715), (5700618.325715, 1682330.705715), (5700687.297145, 1682989.054285), (5700736.77999
9999, 1682983.34), (5700794.154285001, 1682976.31143), (5700932.76857, 1682958.902855), (5700968.13143, 1682954.46),
(5701064.9, 1682942.305715)]]}}
```

developmentSEED

Shapely (Manipulation)

pip install shapely

« If you enjoy and profit from idiomatic Python, appreciate packages that do one thing well, and agree that a spatially enabled RDBMS is often enough the wrong tool for your computational geometry job, Shapely might be for you. »

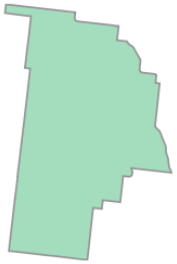
https://shapely.readthedocs.io/en/latest/manual.html

- Good python code
- Good documentation
- Well maintained
- **Shipped with GEOS** (wheels)

https://github.com/Toblerity/Shapely

```
from shapely.geometry import shape
```

```
geom = shape(f["geometry"])
geom
```



```
geom.buffer(5000, cap_style=1)
```

```
c = geom.centroid
print(c)
```

```
POINT (5693345.192108564 1663468.997711802)
```

```
c.buffer(10)
```

```
c.intersects(geom)
```

```
True
```

Geopandas (Analytics)

pip install geopandas

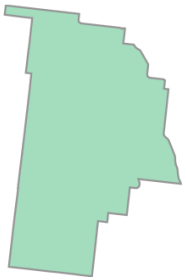
« GeoPandas is a project to add support for geographic data to pandas objects. »

https://github.com/geopandas/geopandas

- Good python code
- Good documentation
- Well maintained

```
import fiona
import geopandas as gpd
import pandas as pd
from shapely.geometry import shape
```

```
pop = pd.read_csv("population.csv")
geoms = gpd.read_file("my-shape.shp")
geoms.loc[0].geometry
```



```
for ind, item in geoms.iterrows():
    row = pop.loc[pop.GEO_CODE == int(item.DAUID)]
    if len(row) != 1:
        continue
    r = int(row.iloc[0][11])
    d = r / item.geometry.area
```

```
print(item)
print(item.geometry.area)
print(r)
print(d)
```

```
DAUID                                                46150068
PRUID                                                      46
PRNAME                                               Manitoba
geometry    POLYGON ((5553387.9 1652727, 5555548.362855 16...
Name: 4656, dtype: object
845293573.8223829
587
6.944332929749011e-07
```

https://github.com/geopandas/geopandas

development SEED

dask (scale)

pip install dask[complete]

« Dask provides advanced parallelism for analytics, enabling performance at scale for the tools you love. »

https://dask.org

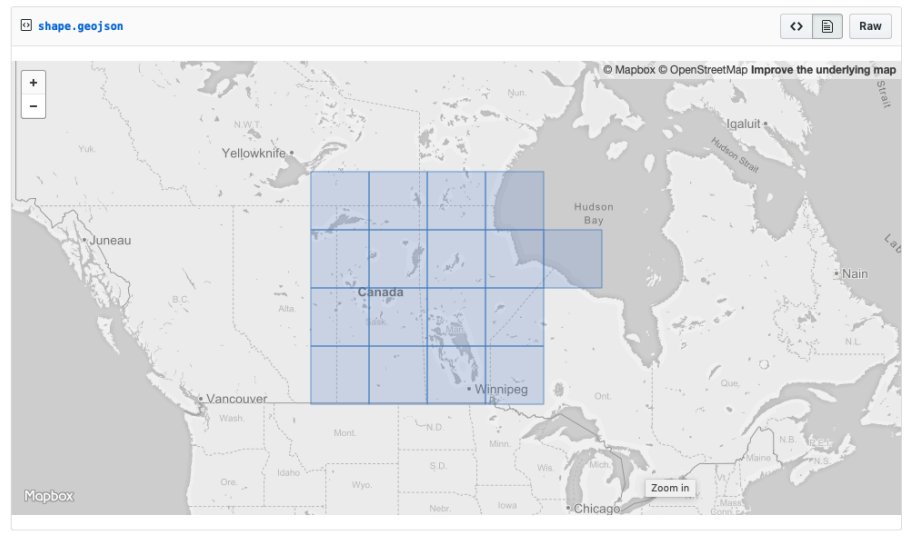
- Pandas integration
- Work at scale

https://github.com/geopandas/geopandas

Others

- scikit-learn (classification / clustering)
- Supermercado
- Mercantile
- http://geojson.io
- https://gist.github.com

$ cat my-shape.geojson| supermercado burn 6 | mercantile shapes | fio collect | gist -f shape.geojson



developmentSEED

Raster

development SEED

GDAL (I/O)

pip install GDAL

« GDAL is a translator library for raster and vector geospatial data formats. »

https://www.gdal.org

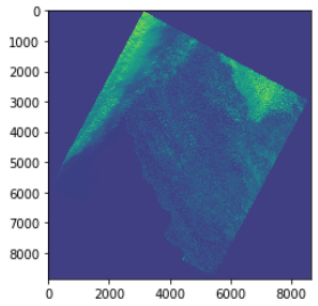
- Not user friendly
- Lack of documentation
- Low level access to GDAL api

```
from osgeo import gdal
%pylab inline

ds = gdal.Open("my-raster.tif")
print(ds.GetProjection())
print(ds.GetGeoTransform())
print()
print(ds.RasterCount, ds.RasterXSize, ds.RasterYSize)
arr = ds.GetRasterBand(1).ReadAsArray()
ds = None
```

```
Populating the interactive namespace from numpy and matplotlib
PROJCS["Albers",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378140,298.2569999999957,A
UTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0],UNIT["degree",0.01745
32925199433],AUTHORITY["EPSG","4326"]],PROJECTION["Albers_Conic_Equal_Area"],PARAMETER["standa
rd_parallel_1",29.5],PARAMETER["standard_parallel_2",45.5],PARAMETER["latitude_of_center",23],
PARAMETER["longitude_of_center",-96],PARAMETER["false_easting",0],PARAMETER["false_northing",0
],UNIT["metre",1,AUTHORITY["EPSG","9001"]]]
(-2335815.0, 30.0, 0.0, 1867515.0, 0.0, -30.0)

3 8661 8851
```

```
imshow(arr)
```

```
<matplotlib.image.AxesImage at 0x133ad96d8>
```



developmentSEED

Rasterio (I/O +)

pip install rasterio

« High performance, lower cognitive load, cleaner and more transparent code. This is what Rasterio is about. »

https://rasterio.readthedocs.io/en/stable/intro.html

- Better python code
- Better documentation
- Written in Cython
- Heavily maintained
- **Shipped with GDAL** (wheels)
- Plugins

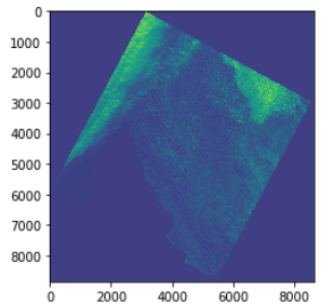
https://github.com/mapbox/rasterio

```
import rasterio
%pylab inline

with rasterio.open("my-raster.tif") as src_dst:
    print(src_dst.meta)
    arr = src_dst.read()
```

```
Populating the interactive namespace from numpy and matplotlib
{'driver': 'GTiff', 'dtype': 'int16', 'nodata': None, 'width': 8661, 'height': 8851, 'count':
3, 'crs': CRS.from_wkt('PROJCS["Albers",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",637
8140,298.2569999999957,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",
0],UNIT["degree",0.0174532925199433],AUTHORITY["EPSG","4326"]],PROJECTION["Albers_Conic_Equal_
Area"],PARAMETER["standard_parallel_1",29.5],PARAMETER["standard_parallel_2",45.5],PARAMETER["
latitude_of_center",23],PARAMETER["longitude_of_center",-96],PARAMETER["false_easting",0],PARA
METER["false_northing",0],UNIT["metre",1,AUTHORITY["EPSG","9001"]]]'), 'transform': Affine(30.
0, 0.0, -2335815.0,
       0.0, -30.0, 1867515.0)}
```

```
imshow(arr[0])
```

```
<matplotlib.image.AxesImage at 0x114265128>
```



developmentSEED

scipy (processing)

pip install scipy scikit-image

« scikit-image is a collection of algorithms for image processing. »

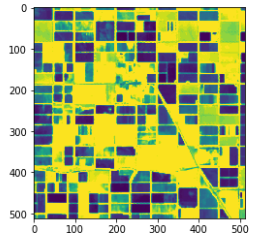
https://scikit-image.org

```
%pylab inline

from skimage import filters
import rasterio
```

```
Populating the interactive namespace from numpy and matplotlib
```

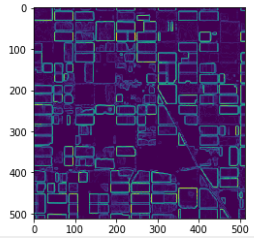
```
file = "1649@2x.png"

with rasterio.open(file) as src_dst:
    image = src_dst.read(indexes=(1))

imshow(image)
```

```
<matplotlib.image.AxesImage at 0x11a1f43c8>
```




```
edges = filters.sobel(image)
imshow(edges)
```

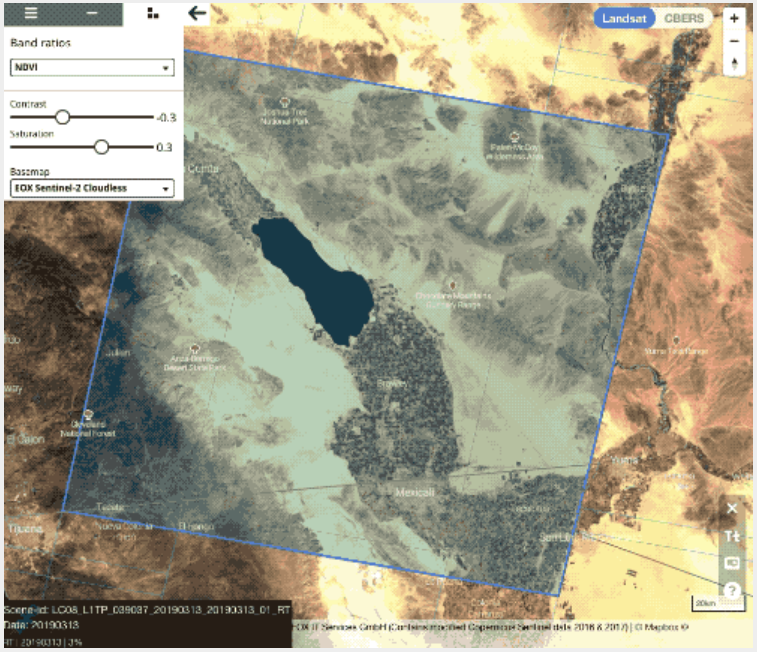
```
<matplotlib.image.AxesImage at 0x11a305dd8>
```



developmentSEED

Others

- xarray
- pillow
- opencv
- rio-tiler



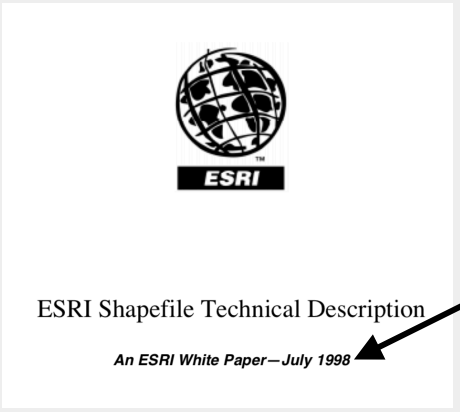
developmentSEED

Unsolicited advices

development SEED

Data format

- **Don’t use shapefile**
- Use **Geojson**
- Save raster as **Cloud Optimized GeoTIFF** (rio-cogeo)
- Checkout PostGIS



developmentSEED

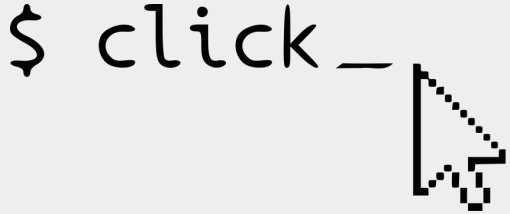
General Python tips

- Use **python3**
- Use **Virtualenvs**
- Write **docstrings**
- Use a **linter** (*pylint, flake8*)
- **tests** your code
- use **CI** (*travis, circle-ci*)
- Use **pre-commit**
- **Open source** everything

My favorite Python modules

- Click (CLI)
- FastAPI (web)
- Geopandas
- Mock
- mapboxgl-jupyter

FastAPI



mapbox

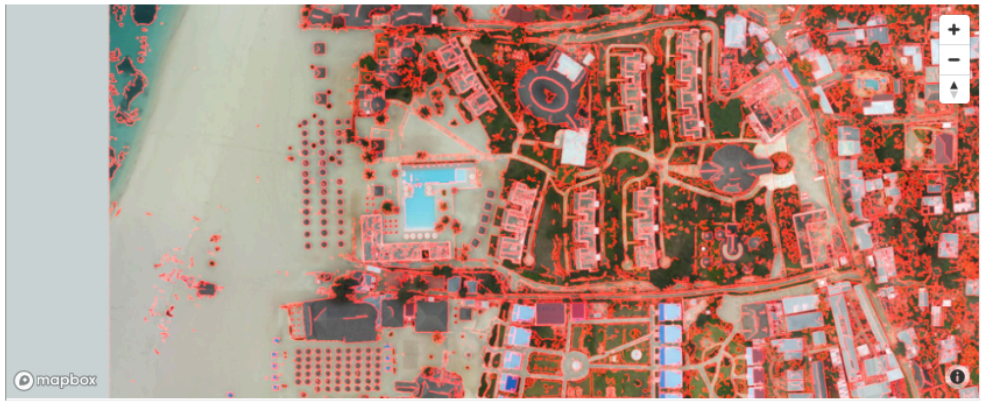
developmentSEED

mapboxgl-jupyter

demo

```
raster = customRaster(file, indexes=(1,2,3))
ts = TileServer(raster)
ts.start()
```

```
viz = RasterTilesViz(ts.get_tiles_url(),
                     tiles_bounds=ts.get_bounds(),
                     center=ts.get_center(),
                     access_token=token,
                     height='400px',
                     zoom=13)
viz.show()
```



developmentSEED

Thanks

development SEED

Links

- https://github.com/datapink/robosat.pink
- https://github.com/developmentseed/label-maker
- https://github.com/sat-utils
- https://github.com/cogeotiff

- http://jwass.github.io/maptime-boston-python/slides
- https://carsonfarmer.com/2013/07/essential-python-geo-libraries/
- https://github.com/giswqs/python-geospatial
- https://geohackweek.github.io/vector/
- https://github.com/pyviz/pyviz
- https://jakevdp.github.io/PythonDataScienceHandbook/index.html
- https://gist.github.com/vincentsarago/ffe44194990b77ca88a923b0db231bfe

development SEED

@_VincentS_

www.developmentseed.org