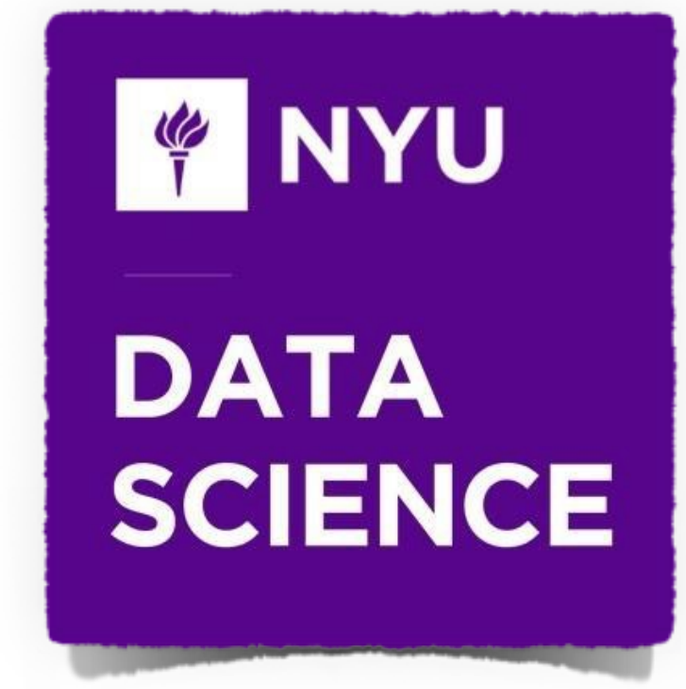
https://github.com/bmtgoncalves/LDSSS17

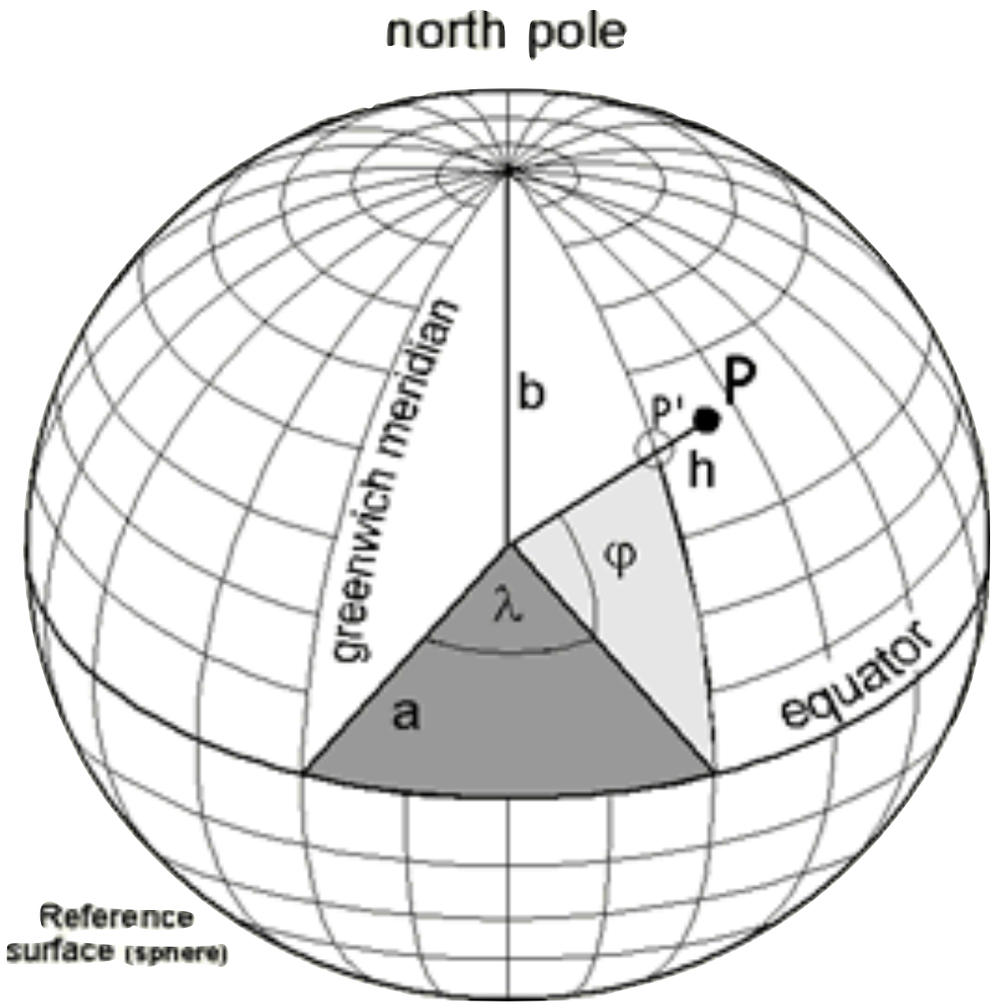
Lviv Data Science Summer School 2017
Mining Geolocated Datasets

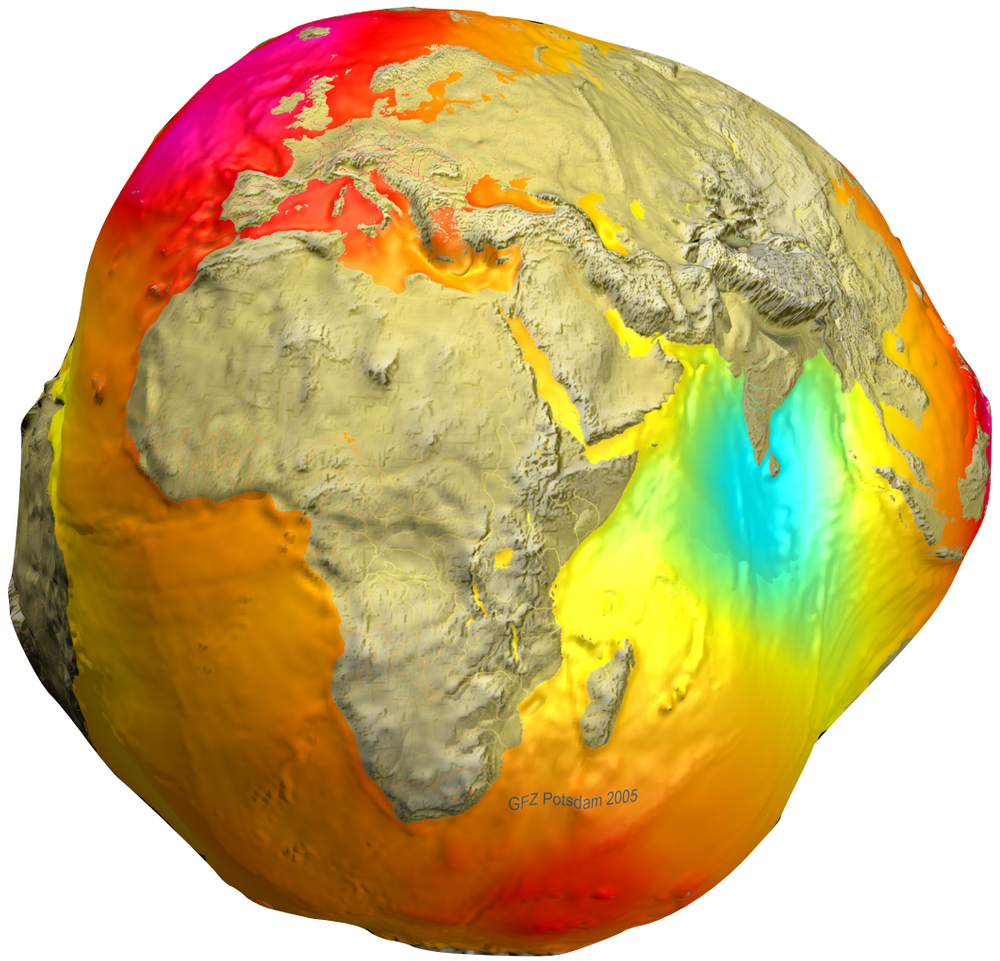
Bruno Gonçalves
www.bgoncalves.com



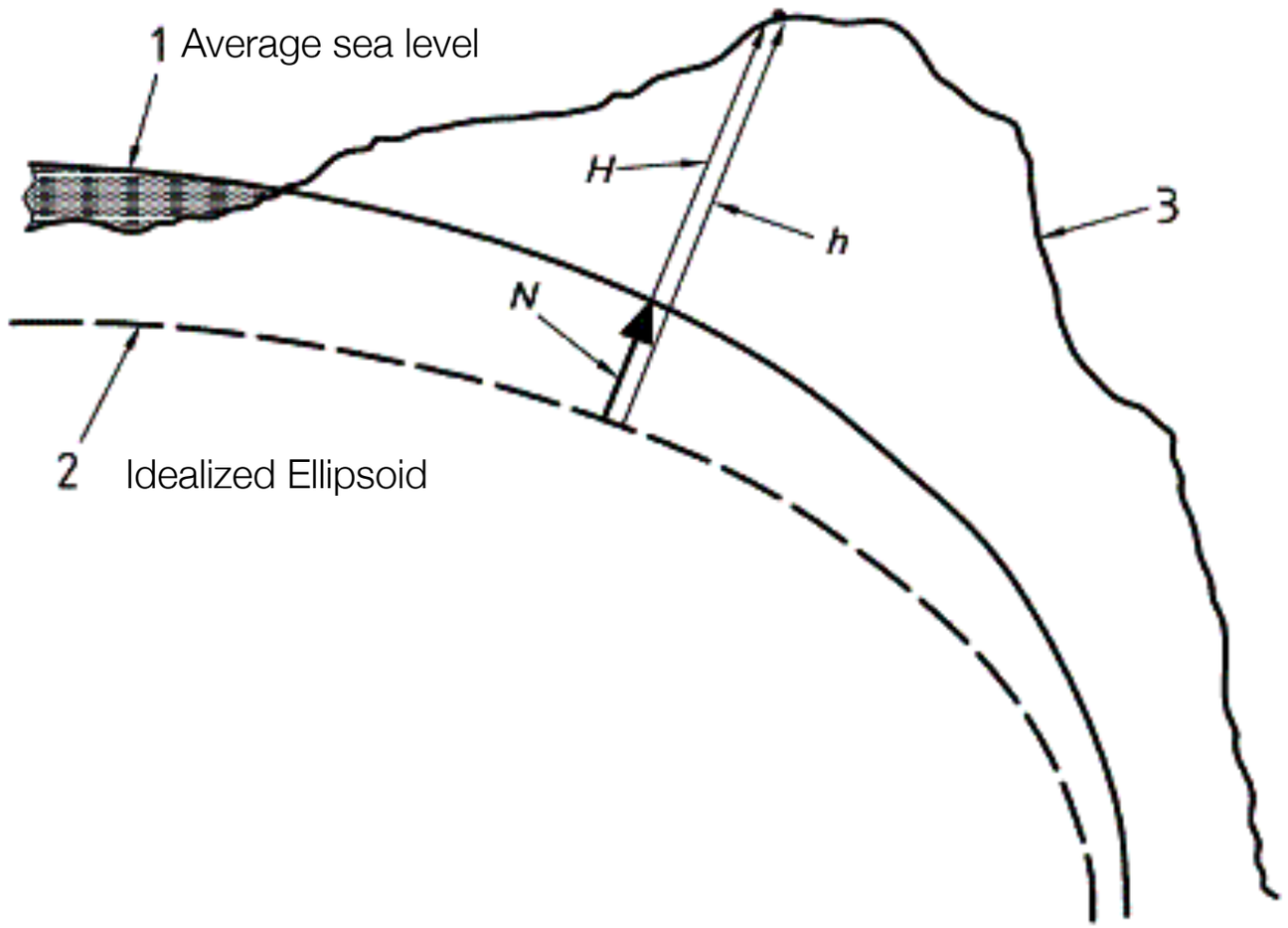
Geographical Information Systems

Global Positioning System (GPS)





Global Positioning System (GPS)



Key

1 geoid
2 ellipsoid
3 surface of the Earth

h = ellipsoidal height, measured from ellipsoid along perpendicular passing through point; $h = H + N$
H = gravity-related height, measured along direction of gravity from vertical datum plane at geoide
N = geoid height, height of geoid above ellipsoid

WGS84

http://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf

- "WGS 84 is an Earth-centered, Earth-fixed terrestrial reference system and geodetic datum"

Coordinate System: Cartesian Coordinates (X, Y, Z). WGS 84 (G1674) follows the criteria outlined in the International Earth Rotation Service (IERS) Technical Note 21. The WGS 84 Coordinate System origin also serves as the geometric center of the WGS 84 Ellipsoid and the Z-axis serves as the rotational axis of this ellipsoid of revolution. WGS 84 geodetic coordinates are generated by using its reference ellipsoid.

Defining Parameters: WGS 84 identifies four defining parameters. These are the semi-major axis of the WGS 84 ellipsoid, the flattening factor of the Earth, the nominal mean angular velocity of the Earth, and the geocentric gravitational constant as specified below.

Parameter	Notation	Value
Semi-major Axis	a	6378137.0 meters
Flattening Factor of the Earth	1/f	298.257223563
Nominal Mean Angular Velocity of the Earth	ω	7292115×10^{-11} radians/second
Geocentric Gravitational Constant (Mass of Earth's Atmosphere Included)	GM**	$3.986004418 \times 10^{14}$ meter^3/second^2

**The value of GM for GPS users is 3.9860050×10^{14} m^3/sec^2 as specified in the references below.

GIS Data Systems

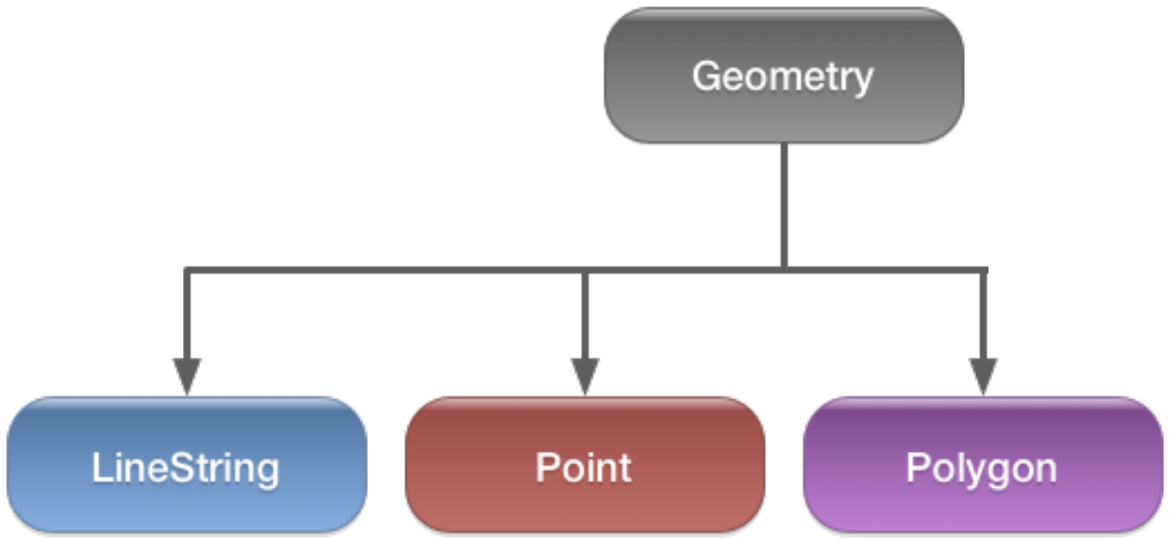
Vector

- Data types:
  - Points
  - Lines
  - Polygons
- Discrete shapes and boundaries
- Spatial, Database and Network analysis
- Shapefile, GeoJSON, GML, etc…

Raster

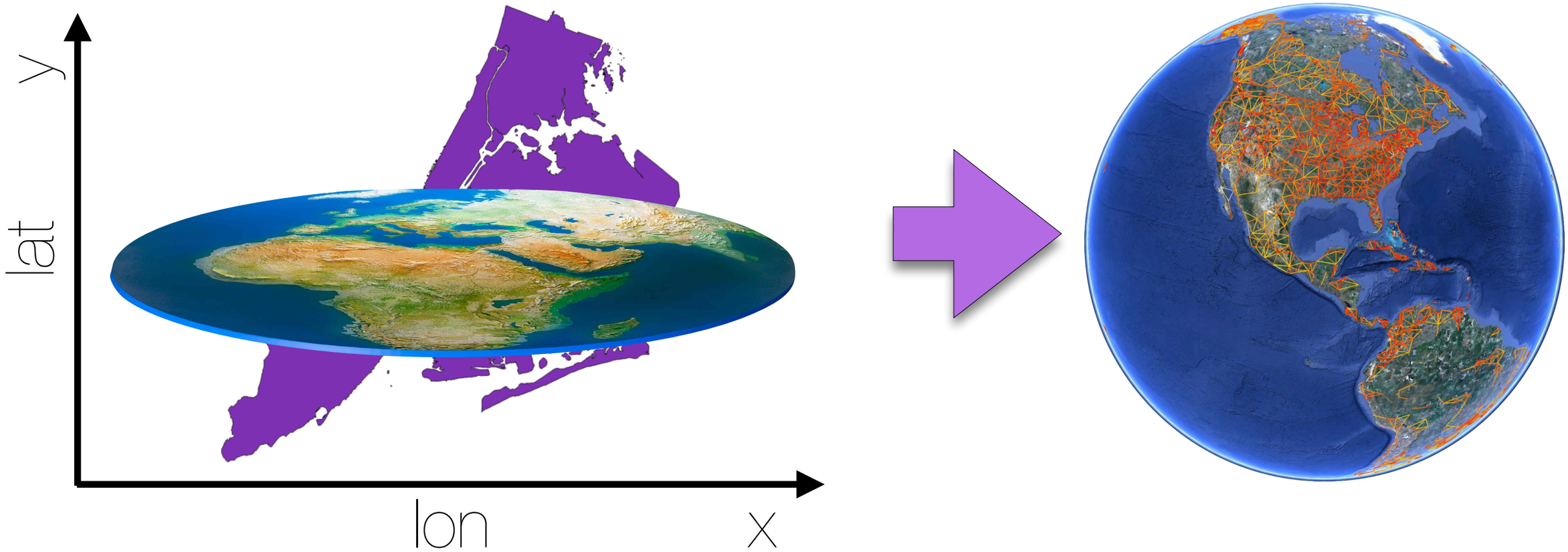
- Data types:
  - Cells
  - Pixels
  - Elements
- Dense data, Continuous surfaces
- Spatial Analysis and modeling
- GeoTIFF, ASC, JPEG2000, etc…

GIS Vector Data types



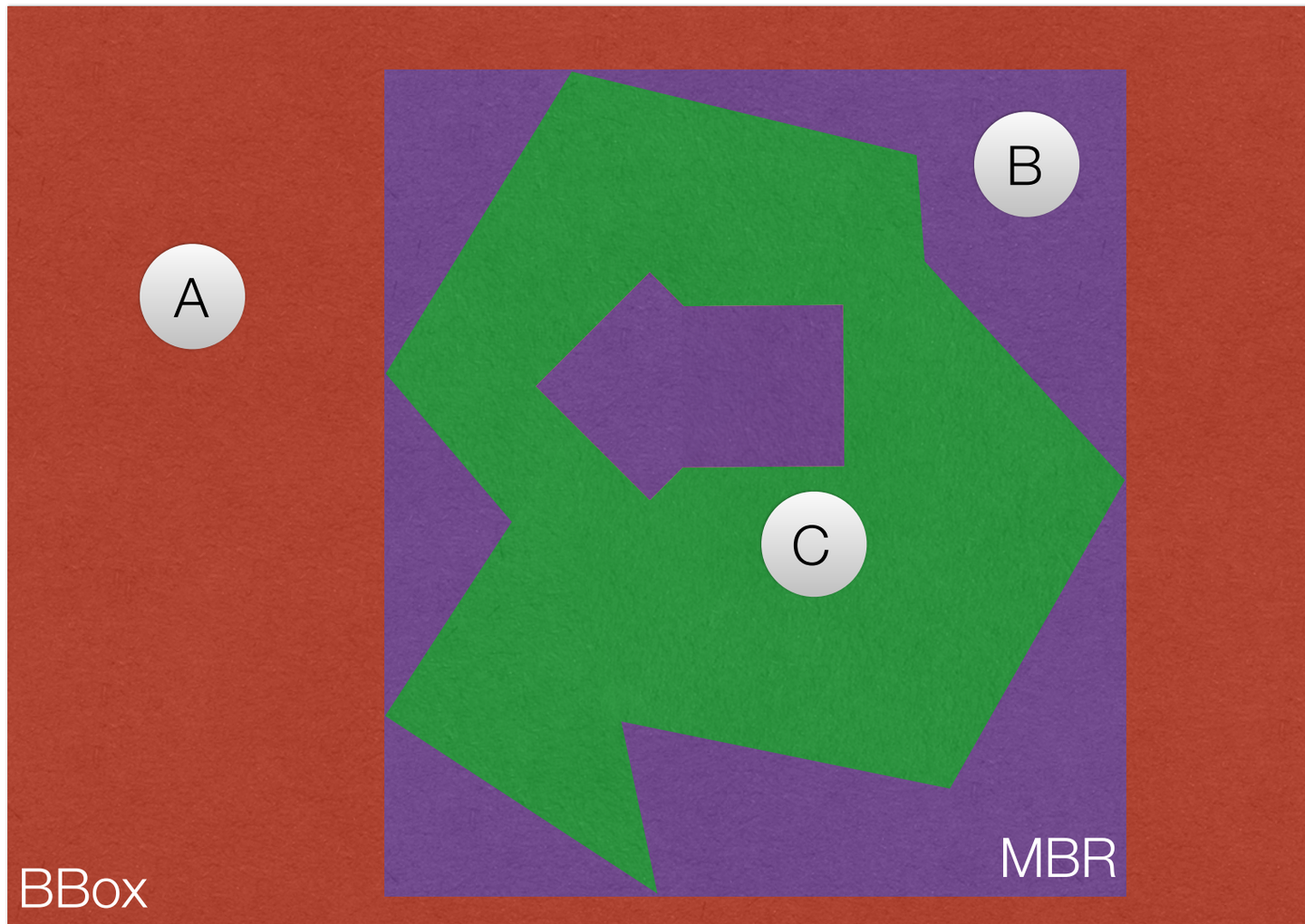
Geographic Information System (GIS)

- Global Positioning System (GPS) - > Latitude, Longitude



Fundamental Concepts

- **Bounding box** - A rectangular box containing the shape of interest
- **Minimum Bounding Rectangle** - The smallest possible Bounding Box that still contains the shape
- **Spatial Reference Identifier** - Maps "flat" lat/lon to a curved surface
  - Specially important to measure distances!
- Rectangles make it easy to quickly check relative positions, but lack precision
  - A within BBox but outside MBR
  - B within MBR but outside shape
  - C within shape
- If BBoxes don't overlap, neither do the shapes they contain

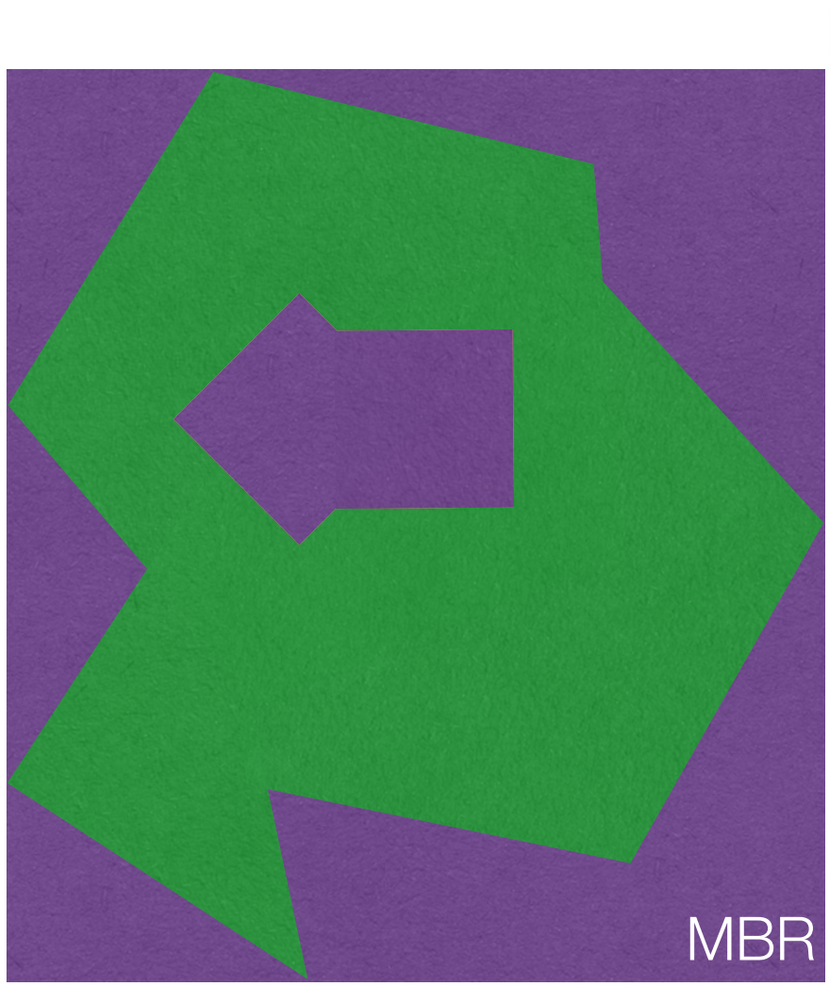


Get MBR/BBox

- You'll often see the term **BBox** to mean the **MBR**

- The MBR can be uniquely defined by the coordinates of two corners.

- Coordinates are simply the extreme values in the four "cardinal" directions, North, South, East and West of all the points defining the shape.



GeoJSON

https://tools.ietf.org/html/rfc7946

- A GeoJSON object is a JSON object specially tailored to spatial data

- It defines

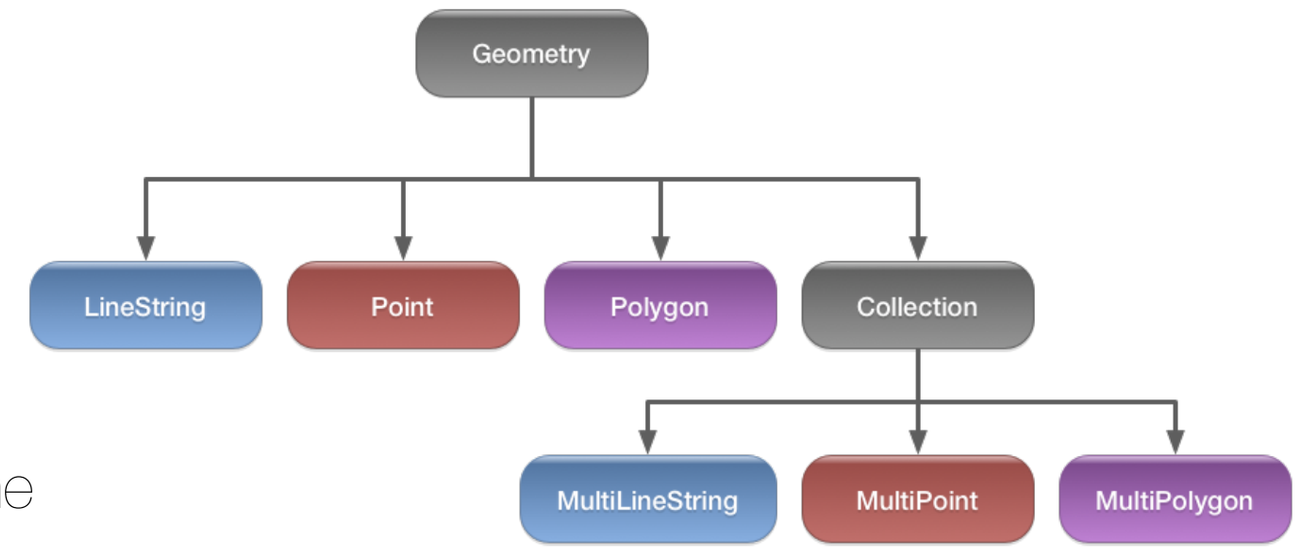
    - Geometry - a region of space

    - Feature - a spatially bounded entity that contains a contains a Geometry object

    - FeatureCollection - A list of Features

- Supports all types of GIS data types:

- Widely supported as an open standard online



GeoJSON

```
['type',
 'crs',
 'features']
```

GeoJSON

```
['type',          'FeatureCollection'
 'crs',           {"properties": {"name": "urn:ogc:def:crs:EPSG::4258"},
 'features']       "type": "name"}
```

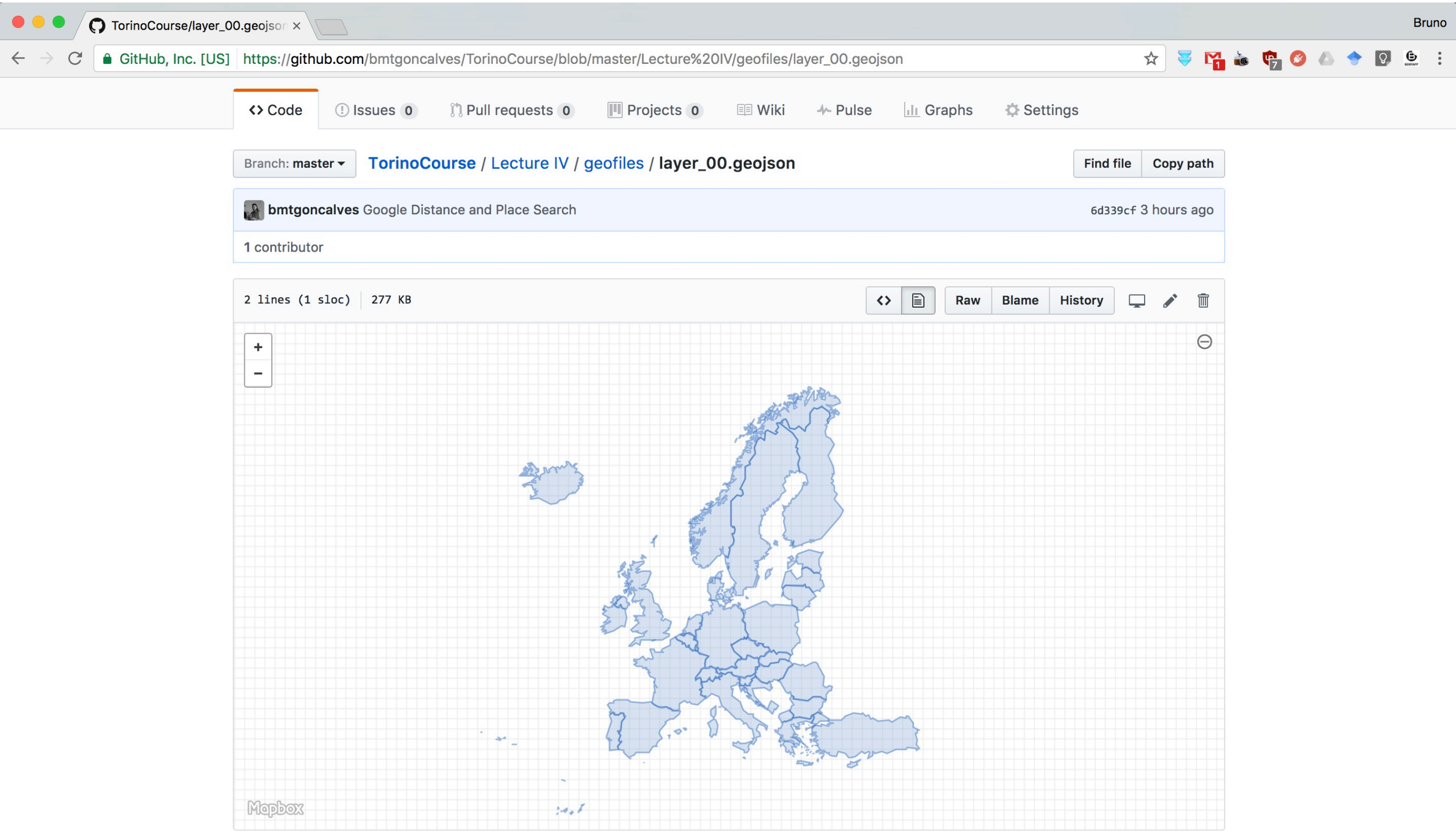
GeoJSON

```
['type',
 'crs',
 'features']   {'geometry': {'coordinates': [[[[19.224069, 43.527541],
                                               [19.227058, 43.47903949800002],
                                               (…)
                                               [19.049223, 43.501789001000002],
                                               [19.224069, 43.527541]]]],
                             'type': 'MultiPolygon'},
                'properties': {'NUTS_ID': 'ME',
                               'SHAPE_AREA': 1.50797533788,
                               'SHAPE_LEN': 7.40877787465,
                               'STAT_LEVL_': 0},
               'type': 'Feature'}
```

GeoJSON

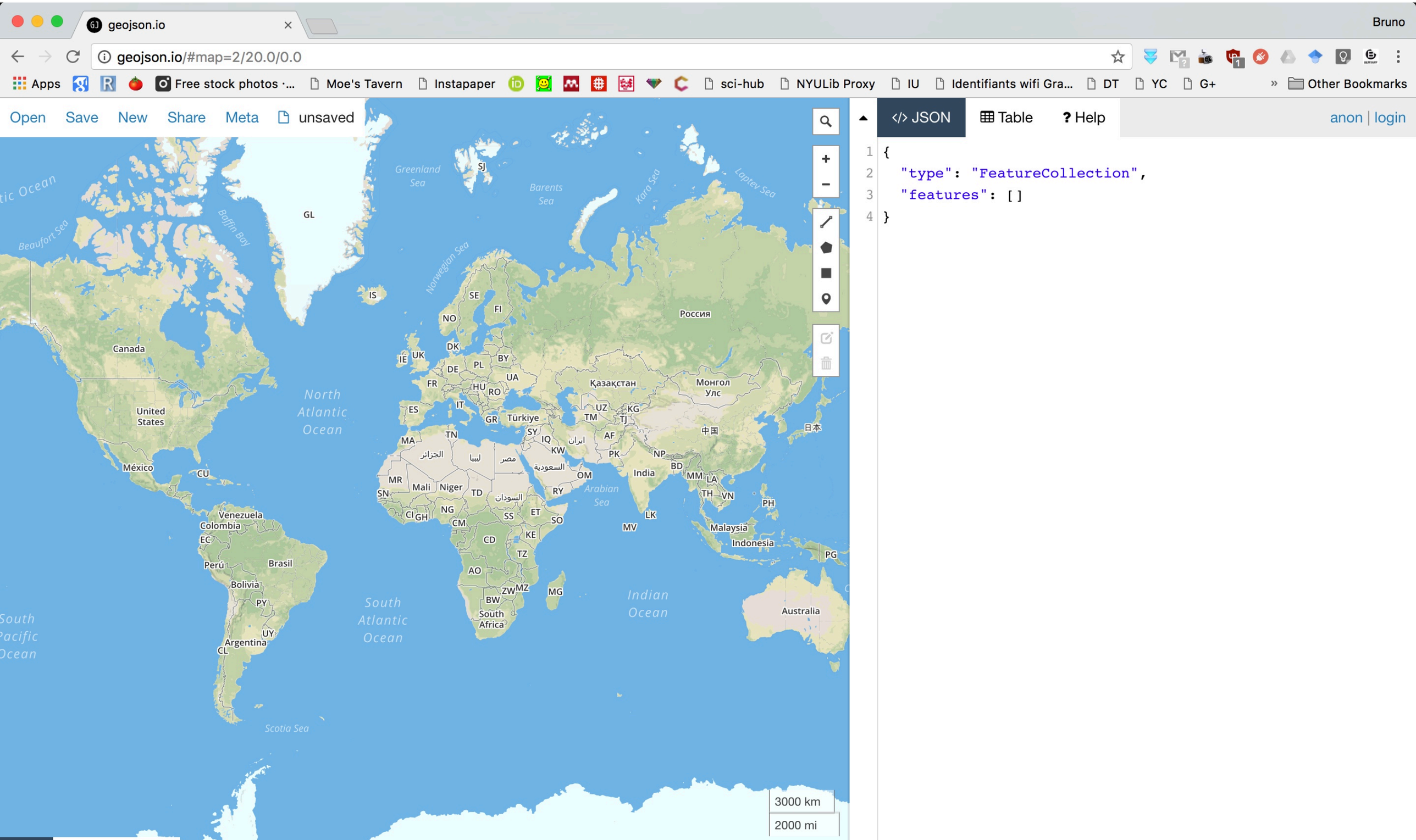
```
['type',
 'crs',
 'features']   {'geometry': {'coordinates': [[[[19.224069, 43.527541],
                                               [19.227058, 43.47903949800002],
                                               (…)
                                               [19.049223, 43.501789001000002],
                                               [19.224069, 43.527541]]]],
                             'type': 'MultiPolygon'},
                'properties': {'NUTS_ID': 'ME',
                               'SHAPE_AREA': 1.50797533788,
                               'SHAPE_LEN': 7.40877787465,
                               'STAT_LEVL_': 0},
               'type': 'Feature'}
```

GitHub support



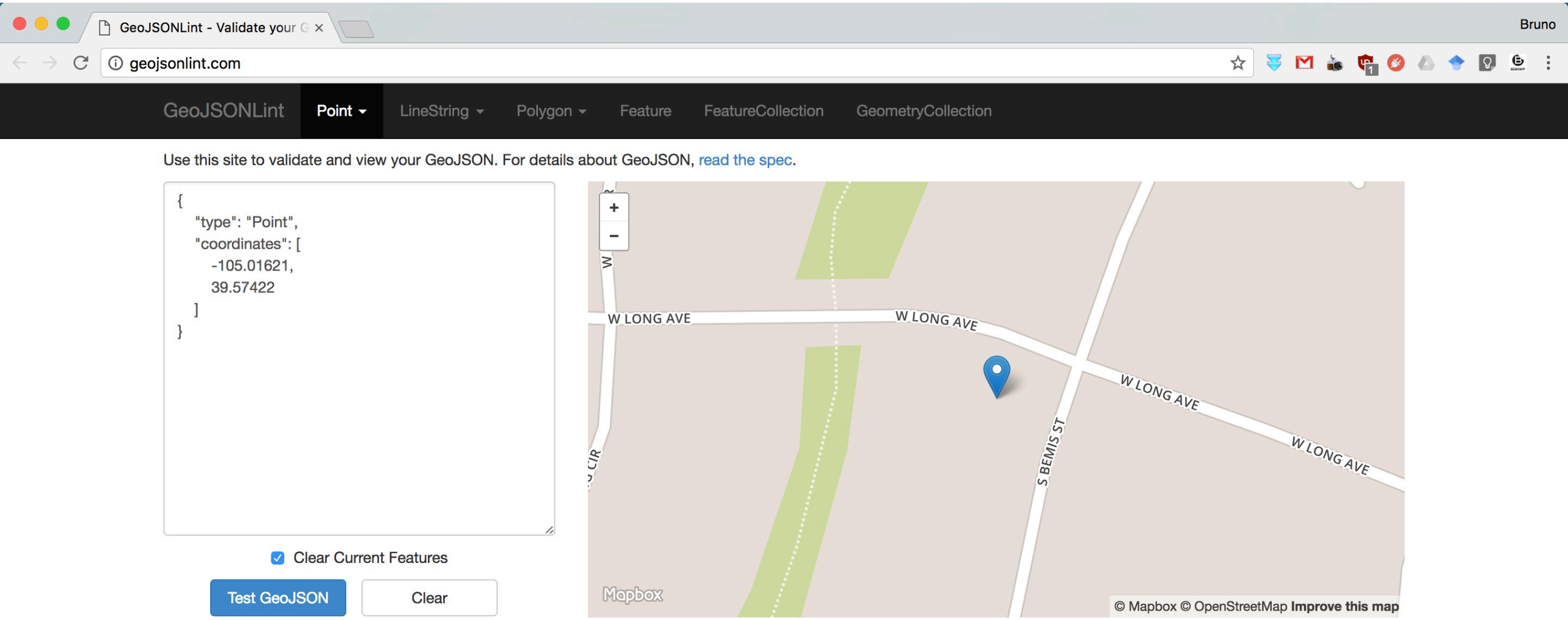
GeoJSON

geojson.io



GeoJSON

http://geojsonlint.com/



Challenge - GeoJSON

- Write a short function to calculate the Bounding Box (MBR) from a GeoJSON file like the one in the **geofiles/** folder

Challenge - GeoJSON

```python
def get_bbox(country):
    maxLat = None
    maxLon = None
    minLat = None
    minLon = None

    for polygon in country["geometry"]["coordinates"]:
        coords = np.array(polygon)[0]

        curMaxLat = np.max(coords.T[1])
        curMinLat = np.min(coords.T[1])

        curMaxLon = np.max(coords.T[0])
        curMinLon = np.min(coords.T[0])

        if maxLat is None or curMaxLat > maxLat:
            maxLat = curMaxLat

        if maxLon is None or curMaxLon > maxLon:
            maxLon = curMaxLon

        if minLat is None or curMinLat < minLat:
            minLat = curMinLat

        if  minLon is None or curMinLon < minLon:
            minLon = curMinLon

    return maxLat, maxLon, minLat, minLon
```

get_bbox.py

Challenge - GeoJSON

```
def plot_country(country):
    for polygon in country["geometry"]["coordinates"]:
        coords = np.array(polygon)

        plt.plot(coords.T[0], coords.T[1])

    maxLat, maxLon, minLat, minLon = get_bbox(country)

    plt.xlim(minLon, maxLon)
    plt.ylim(minLat, maxLat)
```

```
data = json.load(open('geofiles/NUTS_RG_20M_2013.geojson'))

countries = {}

countries["crs"] = data["crs"]
countries["type"] = data["type"]
countries = {}

for feat in data["features"]:
    if feat["properties"]["STAT_LEVL_"] == 0:
        countries[feat["properties"]["NUTS_ID"]] = feat

country = countries["EL"]

plot_country(country)
plt.savefig('Greece.png')
```

get_bbox.py

shapefiles

http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

- Open specification developed by ESRI, still the current leader in commercial GIS software
- shapefiles aren't actua (individual) files…
- but actually a set of files sharing the same name but with different extensions:

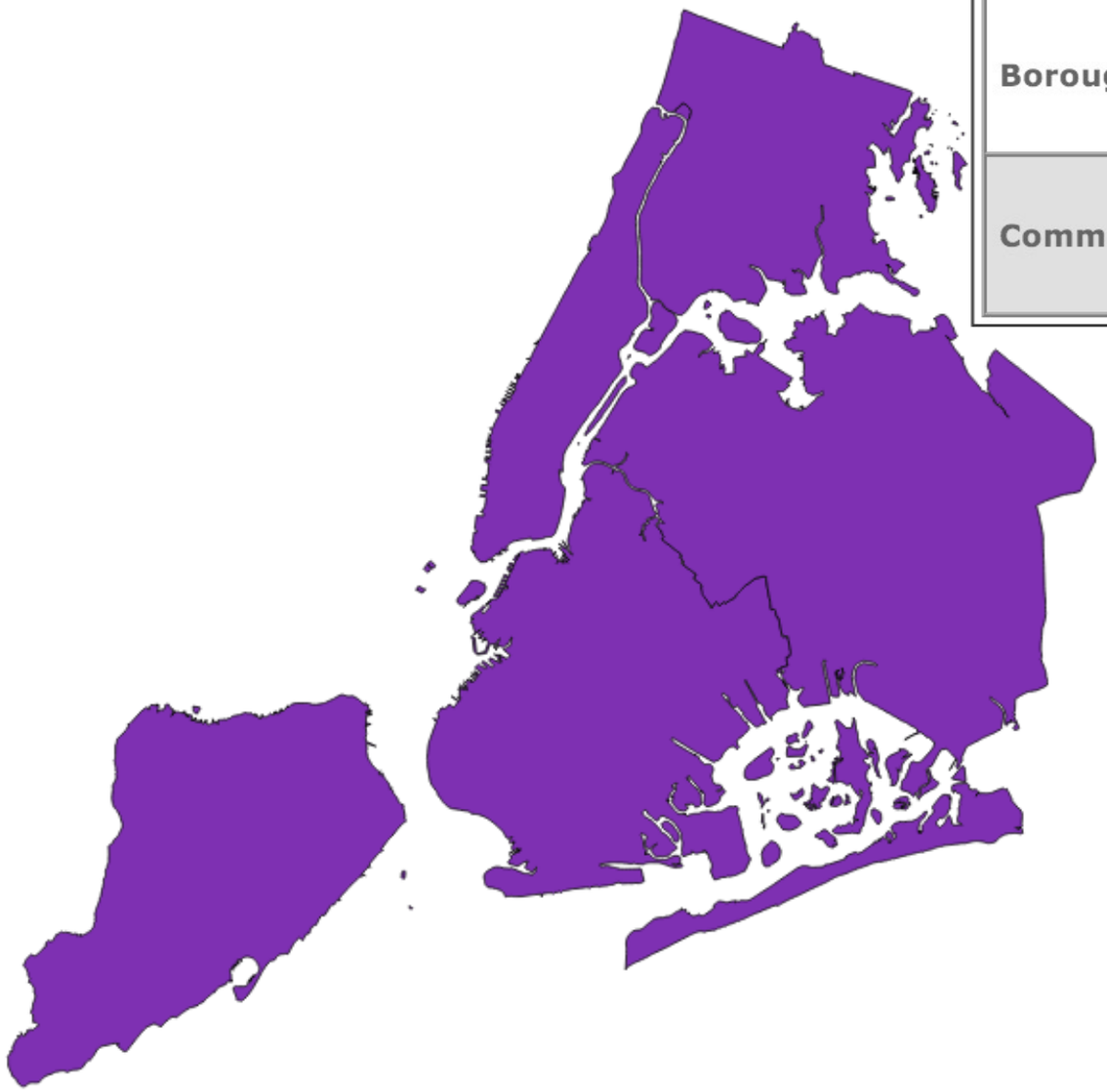
```
(py35) (master) bgoncalves@underdark:$ls -l
total 4856
-rw-r--r--  1 bgoncalves  staff      536 Apr 17 12:40 nybb_wgs84.dbf
-rw-r--r--  1 bgoncalves  staff      143 Apr 17 12:40 nybb_wgs84.prj
-rw-r--r--  1 bgoncalves  staff      257 Apr 17 12:40 nybb_wgs84.qpj
-rw-r--r--  1 bgoncalves  staff  1217376 Apr 17 12:40 nybb_wgs84.shp
-rw-r--r--  1 bgoncalves  staff      140 Apr 17 12:40 nybb_wgs84.shx
(py35) (master) bgoncalves@underdark:$
```

- the actual set of files changes depending on the contents, but three files are usually present:
  - .shp - also commonly referred to as "the" shapefile. Contains the geometric information
  - .dbf - a simple database containing the feature attribute table.
  - .shx - a spatial index, not strictly required

shapefiles

http://www.nyc.gov/html/dcp/html/bytes/districts_download_metadata.shtml#bcd

Borough Boundaries & Community Districts	Download	Metadata
Borough Boundaries (Clipped to Shoreline)	(645k)	
Borough Boundaries (Water Areas Included)	(31k)	
Community Districts (Clipped to Shoreline)	(772k)	



@bgoncalves

QGIS Demo

pyshp

https://github.com/GeospatialPython/pyshp

- pyshp defines utility functions to load and manipulate Shapefiles programmatically.
- The shapefile module handles the most common operations:
  - .Reader(filename) - Returns a Reader object
- Reader.records()/Reader.iterRecords() returns/iterates over the different records present in the shapefile
- Reader.shapes()/Reader.iterShapes() - returns/Iterates over the different shapes present in the shapefile
- Reader.shapeRecords()/Reader.iterShapeRecords() returns/Iterates over both shapes and records present in the shapefile
- Reader.record(index)/Reader.shape(index)/Reader.shapeRecord(index) - return the record/shape/shapeRecord at index position index
- Reader.numRecords - returns the number of records in the shapefile

pyshp

https://github.com/GeospatialPython/pyshp

```
import sys
import shapefile

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

print("Found", shp.numRecords, "records:")

recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))

for record, id in recordDict.items():
    print(id, record)
```


shapefile_load.py

pyshp

https://github.com/GeospatialPython/pyshp
http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

- shape objects contain several fields:
  - bbox - lower left and upper right x,y coordinates (long/lat) - optional
  - parts - list of indexes for the first point of each of the parts making up the shape.
  - points - x,y coordinates for each point in the shape.
  - shapeType - integer representing the shape type - all shapes in a shapefile are required to be of the same shapeType or null.

Value	Shape Type
0	Null Shape
1	Point
3	PolyLine
5	Polygon
8	MultiPoint
11	PointZ
13	PolyLineZ
15	PolygonZ
18	MultiPointZ
21	PointM
23	PolyLineM
25	PolygonM
28	MultiPointM
31	MultiPatch

Challenge - pyshp

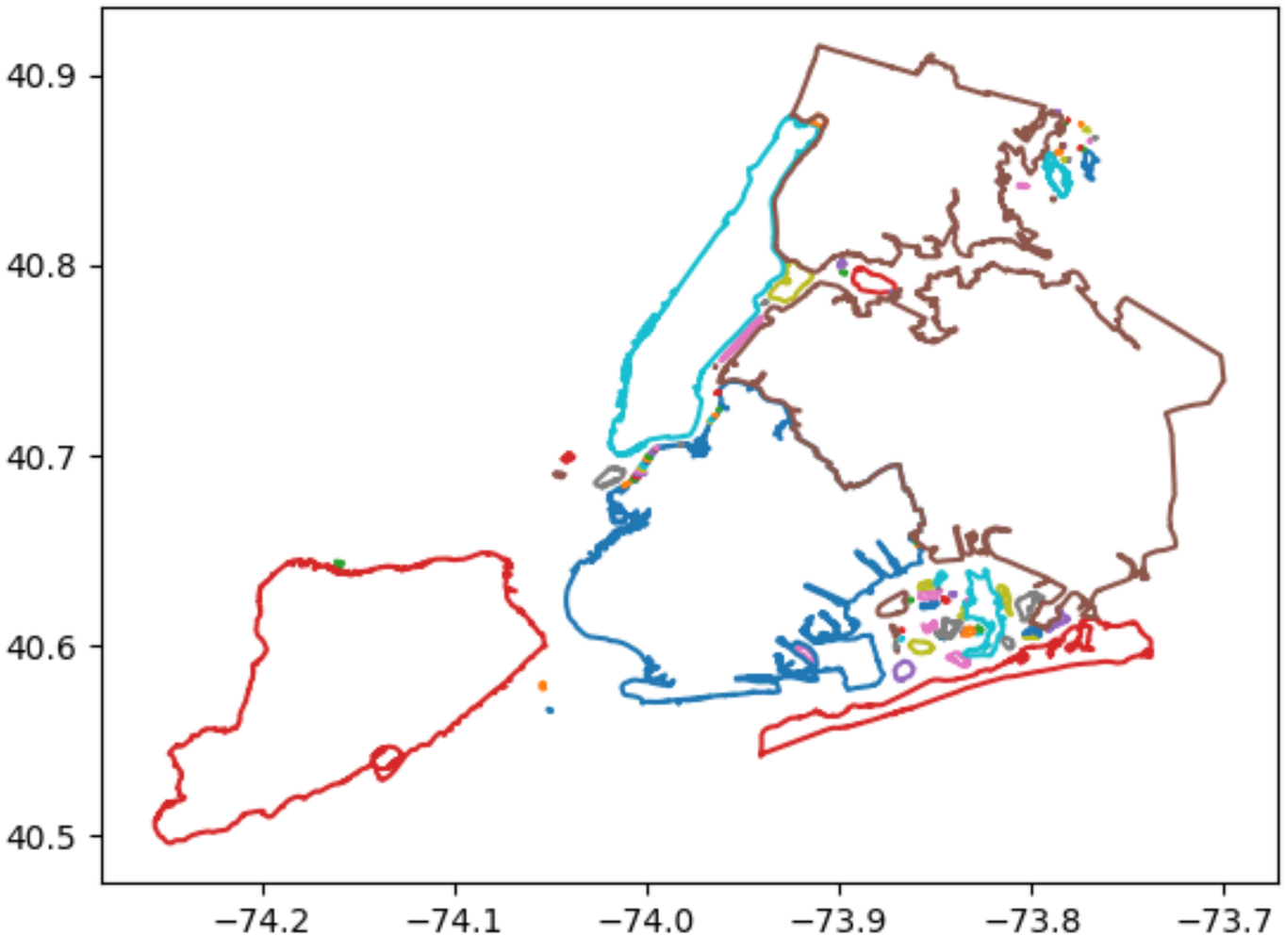
- Write a simple script to plot out all the shapes in:

geofiles/nybb_15c/nybb_wgs84.shp

```
import shapefile
import matplotlib.pyplot as plt
import numpy as np

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

pos = None
count = 0
for shape in shp.iterShapes():
    points = np.array(shape.points)
    parts = shape.parts
    parts.append(len(shape.points))

    for i in range(len(parts)-1):
        plt.plot(points.T[0][parts[i]:parts[i+1]], points.T[1][parts[i]:parts[i+1]])

plt.savefig('NYC.png')
```

plot_shapefile.py

Challenge - pyshp



@bgoncalves

shapely

http://toblerity.org/shapely/manual.html

- Shapely defines geometric objects under shapely.geometry:
  - Point
  - Polygon
  - MultiPolygon
  - shape() Convenience function that creates the appropriate geometric object
- and common operations
  - .crosses(shape) - if it partially overlaps shape
  - .contains(shape) - wether it contains or not the object shape
  - .within(shape)- wether it is contained by object shape
  - .touches(shape) - if the boudaries of this object touch shape

shapely

http://toblerity.org/shapely/manual.html

- shape objects provide useful fields to query a shapes properties:
  - .centroid - The centroid ("center of mass") of the object
  - .area - returns the area of the object
  - .bounds - the MBR of the shape in (minx, miny, maxx, maxy) format
  - .length - the length of the shape
  - .geom_type - the Geometry Type of the object
- shapely.shape is also able to easily load pyshp's shape objects to allow for further manipulations.

shapely

http://toblerity.org/shapely/manual.html

```
import sys
import shapefile
from shapely.geometry import shape

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))

manhattan = shape(shp.shape(recordDict["Manhattan"]))

print("Centroid:", manhattan.centroid)
print("Bounding box:", manhattan.bounds)
print("Geometry type:", manhattan.geom_type)
print("Length:", manhattan.length)
```


shapefile_shape_properties.py

Challenge - Filter points within a Shapefile

- Load each tweet in **NYC.json.gz** file by using:

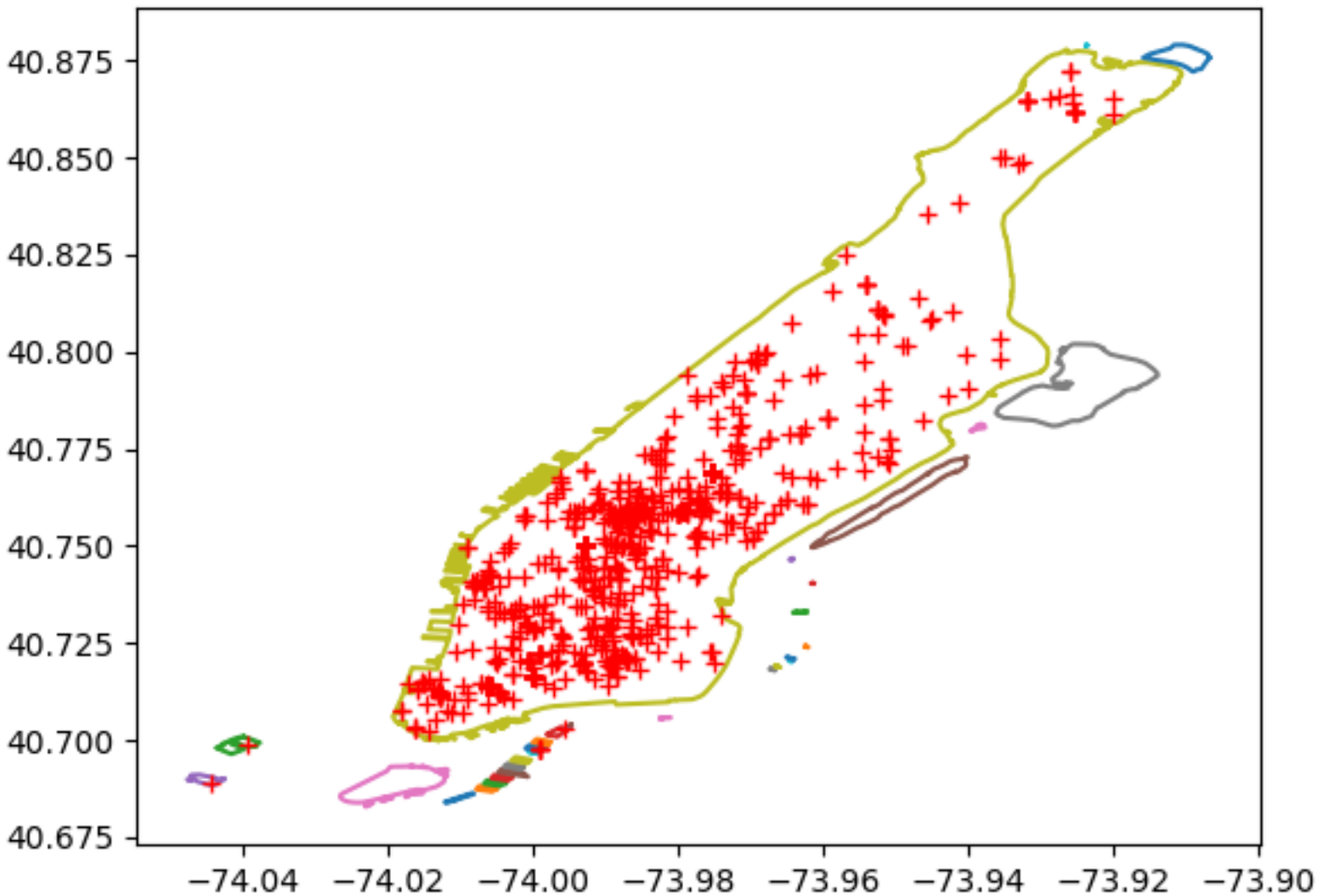
```
tweet = eval(line.strip())
```

-

```
import sys
import shapefile
from shapely.geometry import shape, Point
import gzip

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords())))

manhattan = shape(shp.shape(recordDict["Manhattan"]))
fp = gzip.open("Manhattan.json.gz", "w")

for line in gzip.open("NYC.json.gz"):
    try:
        tweet = eval(line.strip())

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])

            if manhattan.contains(point):
                fp.write(line)
    except:
        pass

fp.close()
```

shapefile_filter.py

Challenge - Filter points within a Shapefile



@bgoncalves

Twitter places

- As we saw last week, Twitter defines a "coordinates" field in tweets
- There is also a "place" field that we glossed over.
- The **place** object contains also geographical information, but at a courser resolution than the **coordinates** field.
- Each place has a unique place_id, a bounding_box and some geographical information, such as **country** and **full_name**:

```
{'attributes': {},
 'bounding_box': {'coordinates': [[[-74.041878, 40.570842],
    [-74.041878, 40.739434],
    [-73.855673, 40.739434],
    [-73.855673, 40.570842]]],
  'type': 'Polygon'},
 'country': 'United States',
 'country_code': 'US',
 'full_name': 'Brooklyn, NY',
 'id': '011add077f4d2da3',
 'name': 'Brooklyn',
 'place_type': 'city',
 'url': 'https://api.twitter.com/1.1/geo/id/011add077f4d2da3.json'}
```

- places can be of several different types: 'admin', 'city', 'neighborhood', 'poi'

Twitter places

- As we saw last week, Twitter defines a "coordinates" field in tweets
- There is also a "place" field that we glossed over.
- The **place** object contains also geographical information, but at a courser resolution than the **coordinates** field.
- Each place has a unique place_id, a bounding_box and some geographical information, such as **country** and **full_name**:

```
{'attributes': {},
 'bounding_box': {'coordinates': [[[-74.041878, 40.570842],
    [-74.041878, 40.739434],
    [-73.855673, 40.739434],
    [-73.855673, 40.570842]]],
  'type': 'Polygon'},
 'country': 'United States',
 'country_code': 'US',
 'full_name': 'Brooklyn, NY',
 'id': '011add077f4d2da3',
 'name': 'Brooklyn',
 'place_type': 'city',
 'url': 'https://api.twitter.com/1.1/geo/id/011add077f4d2da3.json'}
```

The bounding_box field is GeoJSON formatted and compatible with pyshp.shape

- places can be of several different types: 'admin', 'city', 'neighborhood', 'poi'

Twitter places

https://dev.twitter.com/overview/api/places

Place Attributes

Place Attributes are metadata about places. An attribute is a key-value pair of arbitrary strings, but with some conventions.

Below are a number of well-known place attributes which may, or may not exist in the returned data. These attributes are provided when the place was created in the Twitter places database.

Key	Description
street_address	
locality	the city the place is in
region	the administrative region the place is in
iso3	the country code
postal_code	in the preferred local format for the place
phone	in the preferred local format for the place, include long distance code
twitter	twitter screen-name, without @
url	official/canonical URL for place
app:id	An ID or comma separated list of IDs representing the place in the applications place database.

Keys can be no longer than 140 characters in length. Values are unicode strings and are restricted to 2000 characters.

Challenge - Filter points and places

- Load each tweet in NYC.json.gz file by using:

```
tweet = eval(line.strip())
```

- on each line and write to Manhattan_places.json.gz all the tweets within Manhattan, as defined by

```
geofiles/nybb_15c/nybb_wgs84.shp
```

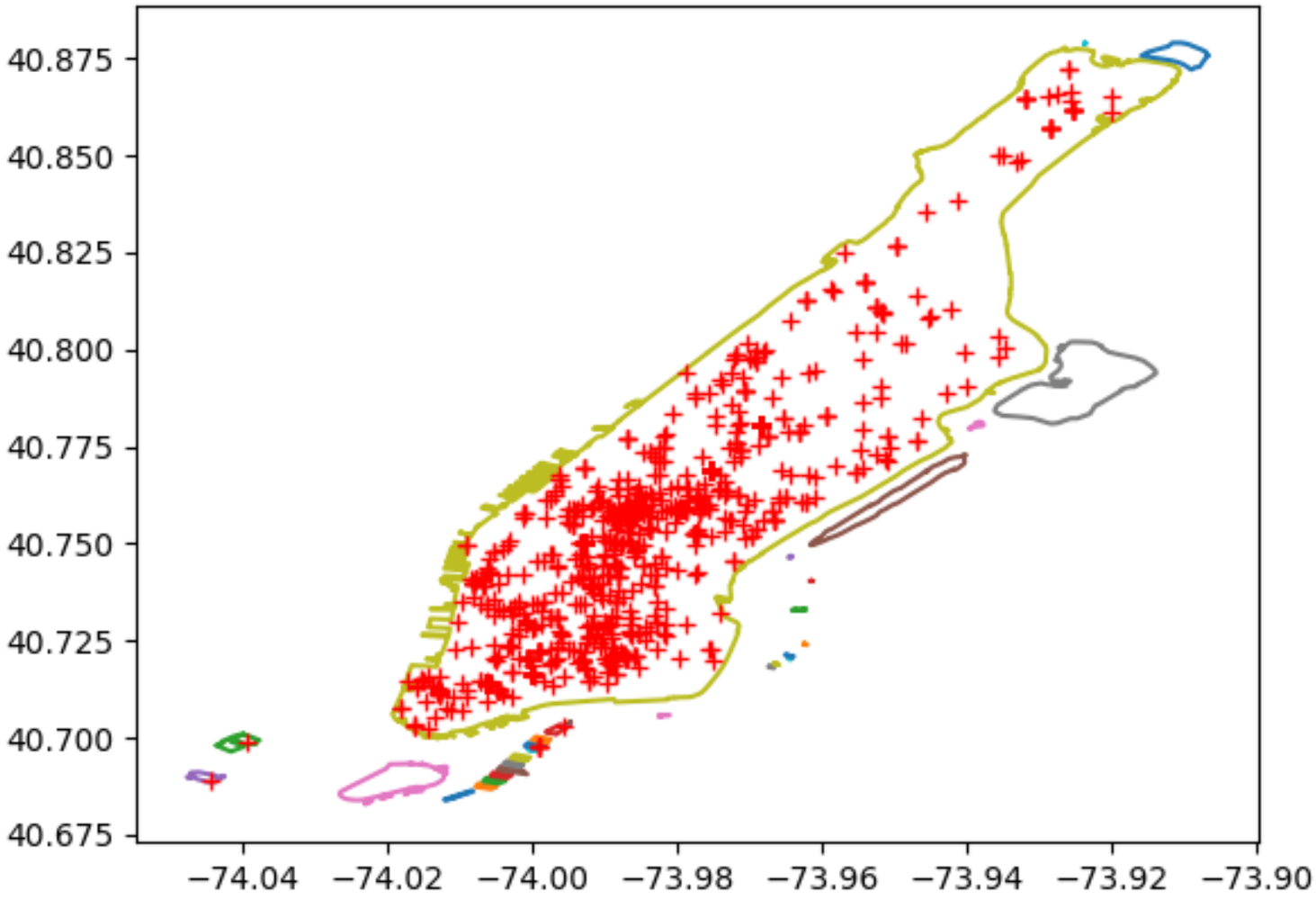
- but now check if the centroid of the 'place' object 'bounding_box' is within Manhattan as well

Challenge - Filter points and places

```
import sys
import shapefile
from shapely.geometry import shape, Point
import gzip

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')

recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))

manhattan = shape(shp.shape(recordDict["Manhattan"]))
fp = gzip.open("Manhattan_places.json.gz", "w")

for line in gzip.open("NYC.json.gz"):
    try:
        tweet = eval(line.strip())
        point = None

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not None:
                bbox = shape(tweet["place"]["bounding_box"])
                point = bbox.centroid

        if point is not None and manhattan.contains(point):
            fp.write(line)
    except:
        pass

fp.close()
```

shapefile_filter_places.py

Challenge - Filter points and places



Challenge - Filter points and places

```
import sys
import gzip
import numpy as np
import shapefile
from shapely.geometry import shape, Point
import matplotlib.pyplot as plt

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')
recordDict = dict(zip([record[1] for record in shp.iterRecords()],
range(shp.numRecords)))

manhattan = shp.shape(recordDict["Manhattan"])

points = np.array(manhattan.points)
parts = manhattan.parts
parts.append(len(manhattan.points))

for i in range(len(parts)-1):
      plt.plot(points.T[0][parts[i]:parts[i+1]], points.T[1]
[parts[i]:parts[i+1]])

points_X = []
points_Y = []

for line in gzip.open(sys.argv[1]):
    try:
        tweet = eval(line.strip())
        point = None

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not
None:
                  bbox = shape(tweet["place"]["bounding_box"])
                  point = bbox.centroid

        if point is not None:
            points_X.append(point.x)
            points_Y.append(point.y)
    except:
        pass

plt.plot(points_X, points_Y, 'r+')

plt.savefig(sys.argv[1] + '.png')
```

@bgoncalves

plot_shapefile_points.py

Calculating distances

https://en.wikipedia.org/wiki/Vincenty%27s_formulae
https://en.wikipedia.org/wiki/Great-circle_distance
https://en.wikipedia.org/wiki/Haversine_formula

- Earlier we saw how to obtain the distance between two points using the Google Maps API.

- But what is the shortest distance between two **arbitrary** points on the surface of the Earth?

- This depends strongly on our model of the Earth:

  - Great Circle - Assumes that the Earth is a perfect sphere of a given radius

    - Usually uses the Haversine formula $\Delta\sigma = 2\arcsin\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)}$

  - Vincenty - Uses a (more) accurate ellipsoid model of the Earth

geopy

https://geopy.readthedocs.io/en/1.10.0/

- geopy provides two different types of functionality
    - geopy.geocoders - a unified interface to several geocoding services (Google Maps, Nominatim, Yahoo, Bing, etc…)
    - geopy.distance - state of the art distance calculations
- We will focus just on the distance module:
    - distance.vincenty(p1, p2) - Calculate the vincenty distance between p1 and p2
    - distance.great_circle(p1, p2) - Calculate the great circle distance between p1 and p2
    - distance.distance(p1, p2) - an alias to distance.vincenty to be used as a default.

geopy

https://geopy.readthedocs.io/en/1.10.0/

- all distance functions return a Distance object.
- the Distance object provides properties that represent the result in different units:
  - .km/.kilometers
  - .m/.meters
  - .mi/.miles
  - .ft/.feet
  - .nm/.nautical
- it also allows us to recalculate the result using different ellipsoids:
  - .set_ellipsoid('elliipsoid')
  - by default WGS-84 is used.

```
                 model             major (km)   minor (km)     flattening
ELLIPSOIDS = {'WGS-84':        (6378.137,    6356.7523142,  1 /
              'GRS-80':        (6378.137,    6356.7523141,  1 /
              'Airy (1830)':   (6377.563396, 6356.256909,   1 /
              'Intl 1924':     (6378.388,    6356.911946,   1 / 297.0),
              'Clarke (1880)': (6378.249145, 6356.51486955, 1 / 293.465),
              'GRS-67':        (6378.1600,   6356.774719,   1 / 298.25),
              }
```

Challenge - geopy

https://geopy.readthedocs.io/en/1.10.0/

- Calculate the distance between

p1 = (41.49008, -71.312796)
p2 = (41.499498, -81.695391)

- in meters, using the **vincenty** and **great_circle** functions.

```
from geopy import distance

p1 = (41.49008, -71.312796)
p2 = (41.499498, -81.695391)

dist_vincenty = distance.vincenty(p1,
p2).meters
dist_great = distance.great_circle(p1,
p2).meters

print("Vincenty:", dist_vincenty)
print("Great Circles:", dist_great)
```

geopy_distance.py

GIS Data Systems

Vector

- Data types:
  - Points
  - Lines
  - Polygons
- Discrete shapes and boundaries
- Spatial, Database and Network analysis
- Shapefile, GeoJSON, GML, etc…

Raster

- Data types:
  - Cells
  - Pixels
  - Elements
- Dense data, Continuous surfaces
- Spatial Analysis and modeling
- GeoTIFF, ASC, JPEG2000, etc…

ASCII Grid

- Perhaps the simples raster file
- ASCII text based
- A small header

```
ncols         246
nrows         119
xllcorner     -126.500000000000
yllcorner     22.750000000000
cellsize      0.250000000000
NODATA_value  -9999
```

- Followed by rows of numbers
- Very convenient to Read and Write

geofiles/US_pop.asc

Challenge - ASCII Grid

- Write a simple function to load the file:

geofiles/US_pop.asc

- into a numpy array and print the value corresponding to:

40.730503, -74.243251

Challenge

```python
import numpy as np
import matplotlib.pyplot as plt

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y

fp = open("geofiles/US_pop.asc")
ncols, count = fp.readline().split()
ncols = int(count)
nrows, count = fp.readline().split()
nrows = int(count)
xllcorner, value = fp.readline().split()
xllcorner = float(value)
yllcorner, value = fp.readline().split()
yllcorner = float(value)
cellsize, value = fp.readline().split()
cellsize = float(value)

NODATA_value, value = fp.readline().split()
NODATA_value = float(value)

data = []
for line in fp:
    fields = line.strip().split()
    data.append([float(field) for field in fields])

data = np.array(data)
data[data==NODATA_value] = 0

x = -74.243251
y = 40.730503

coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, x, y)
print(data[coord_y, coord_x])
```

asc_load.py

Challenge

```python
import numpy as np
import matplotlib.pyplot as plt

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y

fp = open("geofiles/US_pop.asc")
ncols, count = fp.readline().split()
ncols = int(count)
nrows, count = fp.readline().split()
nrows = int(count)
xllcorner, value = fp.readline().split()
xllcorner = float(value)
yllcorner, value = fp.readline().split()
yllcorner = float(value)
cellsize, value = fp.readline().split()
cellsize = float(value)

NODATA_value, value = fp.readline().split()
NODATA_value = float(value)

data = []
for line in fp:
    fields = line.strip().split()
    data.append([float(field) for field in fields])

data = np.array(data)
data[data==NODATA_value] = 0

x = -74.243251
y = 40.730503

coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, x, y)
print(data[coord_y, coord_x])
```

asc_load.py

Challenge - ASCII Grid

asc_load.py

ASCII Grid

- This type of grid is a very convenient way to aggregate spatial data.
- Simply map **lat, lon** pairs to matrix entries and then increment the values
- All we need is to define the **bbox** we are interested in, and the size of each cell and create a matrix with that shape.

```
import numpy as np
from shapely.geometry import shape, Point
import shapefile

shp = shapefile.Reader('../Lecture IV/geofiles/nybb_15c/nybb_wgs84.shp')
recordDict = dict(zip([record[1] for record in shp.iterRecords()],
range(shp.numRecords)))

manhattan = shp.shape(recordDict["Manhattan"])

xllcorner, yllcorner, xurcorner, yurcorner = manhattan.bbox

cellsize = 0.01

ncols = int((xurcorner-xllcorner)/cellsize)
nrows = int((yurcorner-yllcorner)/cellsize)

data = np.zeros((nrows, ncols), dtype='int')

print(data.shape)
```

asc_generate_grid.py

Challenge - Aggregate

- Rewrite

shapefile_filter_places.py

- so that it now counts how many tweets happen in each cell of width 0.01

Challenge - Aggregate

```python
import sys
import numpy as np
import shapefile
from shapely.geometry import shape, Point
import matplotlib.pyplot as plt
import gzip

def map_points(xllcorner, yllcorner, cellsize, nrows, x, y):
    x = int((x-xllcorner)/cellsize)
    y = (nrows-1)-int((y-yllcorner)/cellsize)

    return x, y
```

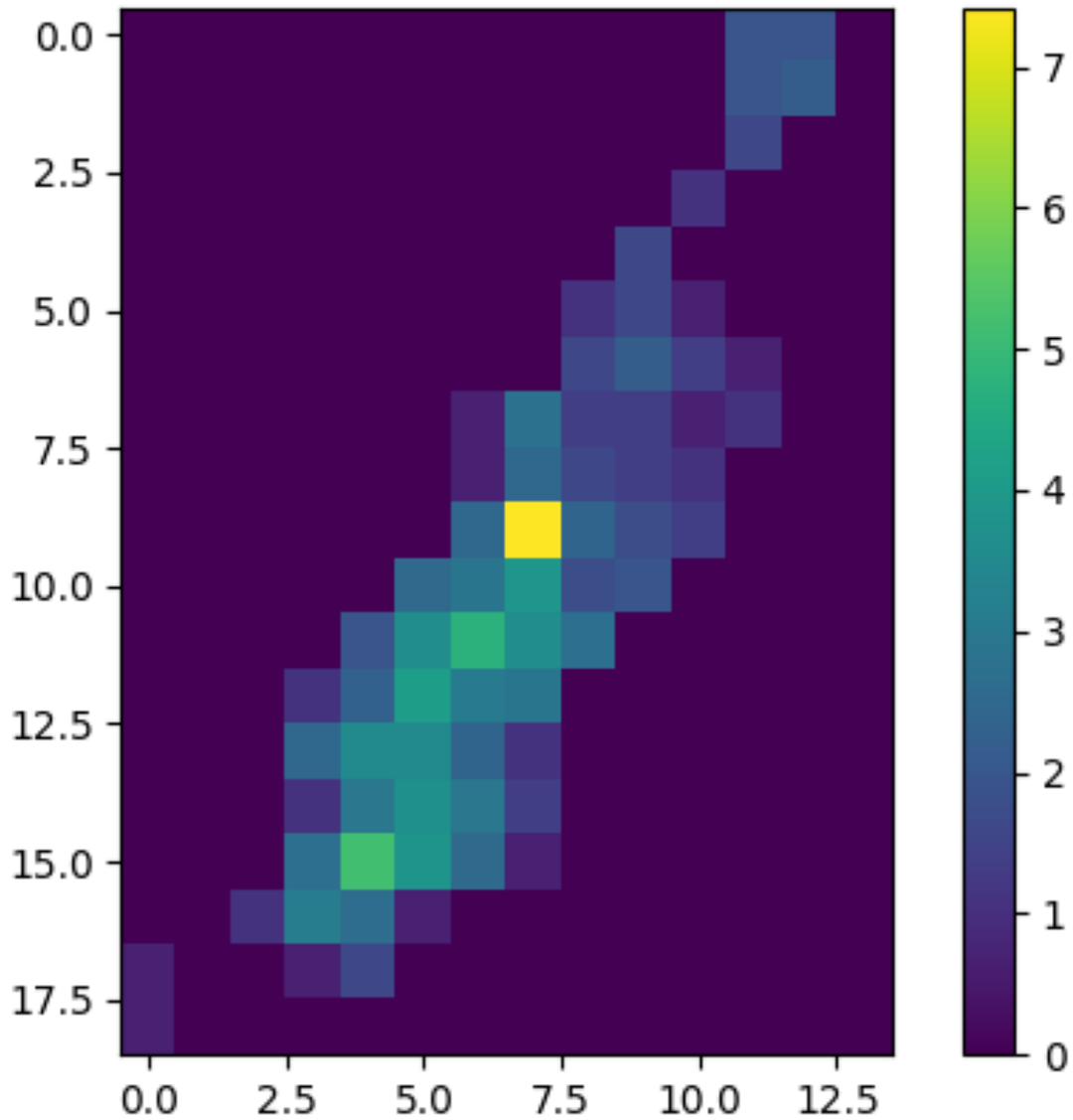
```python
def save_asc(data, xllcorner, yllcorner, cellsize, filename):
    fp = open(filename, "w")

    nrows, ncols = data.shape

    print("ncols", ncols, file=fp)
    print("nrows", nrows, file=fp)
    print("xllcorner", xllcorner, file=fp)
    print("yllcorner", yllcorner, file=fp)
    print("cellsize", cellsize, file=fp)
    print("NODATA_value", "-9999", file=fp)

    for i in range(nrows):
        for j in range(ncols):
            print(("%u " % data[i, j]), end="", file=fp)

        print("\n", end="", file=fp)

    fp.close()
```

shapefile_filter_aggregate.py

```
shp = shapefile.Reader('../Lecture IV/geofiles/nybb_15c/nybb_wgs84.shp')
recordDict = dict(zip([record[1] for record in shp.iterRecords()], range(shp.numRecords)))
manhattan = shape(shp.shape(recordDict["Manhattan"]))

xllcorner, yllcorner, xurcorner, yurcorner = manhattan.bounds
cellsize = 0.01

ncols = int((xurcorner-xllcorner)/cellsize)
nrows = int((yurcorner-yllcorner)/cellsize)

data = np.zeros((nrows, ncols), dtype='int')

for line in gzip.open("../Lecture IV/NYC.json.gz"):
    try:
        tweet = eval(line.strip())
        point = None

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not None:
                bbox = shape(tweet["place"]["bounding_box"])
                point = bbox.centroid

        if point is not None and manhattan.contains(point):
            coord_x, coord_y = map_points(xllcorner, yllcorner, cellsize, nrows, point.x, point.y)
            data[coord_y, coord_x] += 1

    except:
        pass


save_asc(data, xllcorner, yllcorner, cellsize, "Manhattan.asc")

plt.imshow(np.log(data+1))
plt.colorbar()
plt.savefig('Manhattan_cells.png')
```

shapefile_filter_aggregate.py

Challenge - Aggregate

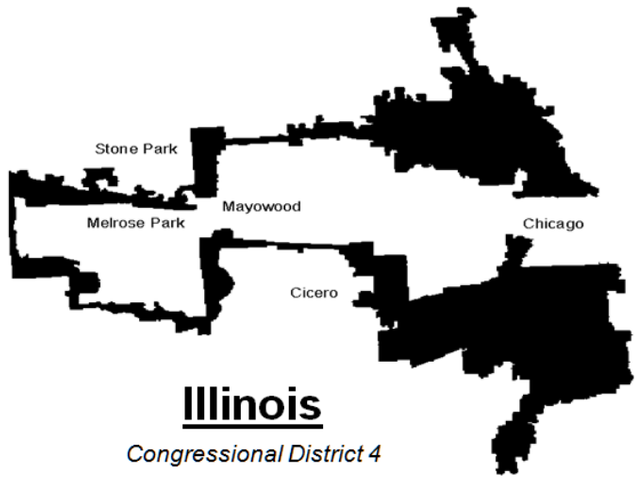


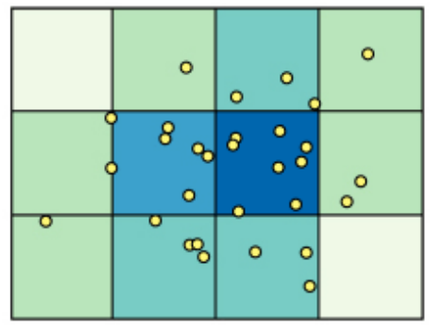
Modifiable Areal Unit Problem

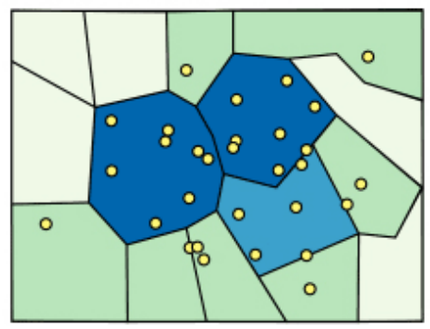
- Different aggregations can result in differing results.

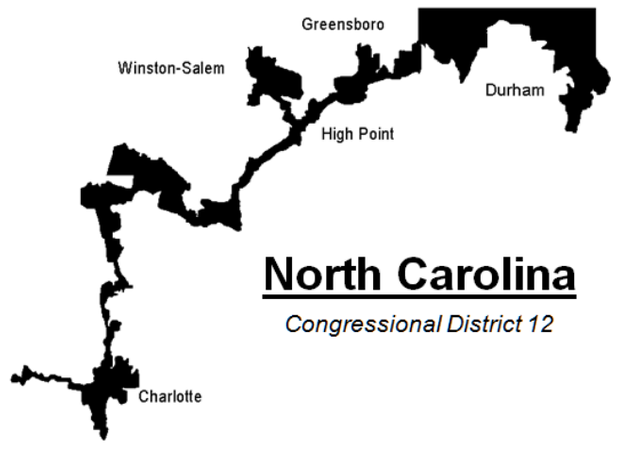
- The aggregatic
  the problem ar
  problem.

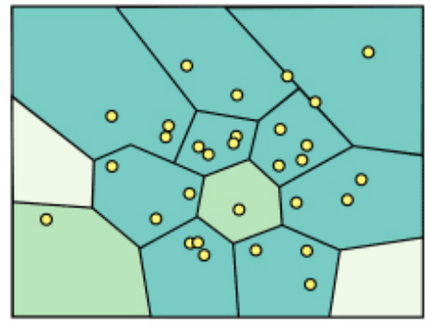
- Sometimes ab











...graphics from 1990s Supreme Court Redistricting Decisions, Peter S. Wattson
www.senate.leg.state.mn.us/departments/scr/REDIST/red907.htm

Challenge - Aggregate Shapefile

- Rewrite

shapefile_filter_aggregate.py

- so that it now counts how many tweets occur in each of the boroughs. Write counts into a csv file

```python
import sys
import gzip
import shapefile
from shapely.geometry import shape, Point
import numpy as np

shp = shapefile.Reader('geofiles/nybb_15c/nybb_wgs84.shp')
shapes = [shape(shp.shape(i)) for i in range(shp.numRecords)]
counts = np.zeros(shp.numRecords, dtype='int')

tweet_count = 0

for line in gzip.open("NYC.json.gz"):
    try:
        tweet = eval(line.strip())
        point = None

        tweet_count += 1

        if tweet_count % 100 == 0:
            print(tweet_count, file=sys.stderr)

        if "coordinates" in tweet and tweet["coordinates"] is not None:
            point = Point(tweet["coordinates"]["coordinates"])
        else:
            if "place" in tweet and tweet["place"]["bounding_box"] is not None:
                bbox = shape(tweet["place"]["bounding_box"])
                point = bbox.centroid

        if point is not None:
            for borogh in range(shp.numRecords):
                if shapes[borogh].contains(point):
                    counts[borogh] += 1
                    break
    except:
        pass
```

@bgoncalves

shapefile_aggregate.py

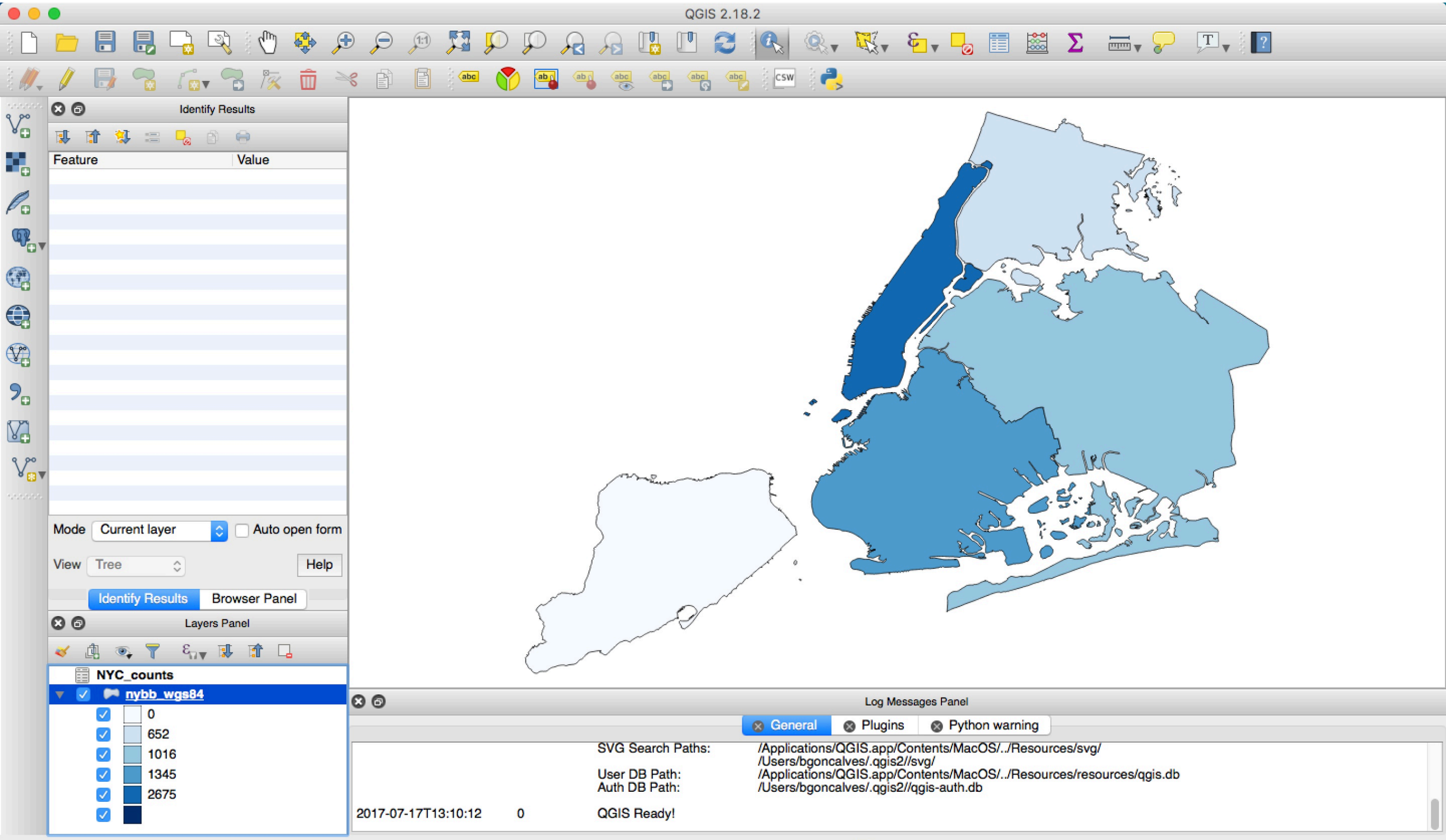
Challenge - Aggregate Shapefile

```
fp_out = open("NYC_counts.csv", "w")

print("id,name,count", file=fp_out)
for borogh in range(shp.numRecords):
    record = shp.record(borogh)
    print(",".join([str(record[0]), record[1], str(counts[borogh])]), file=fp_out)

fp_out.close()
```

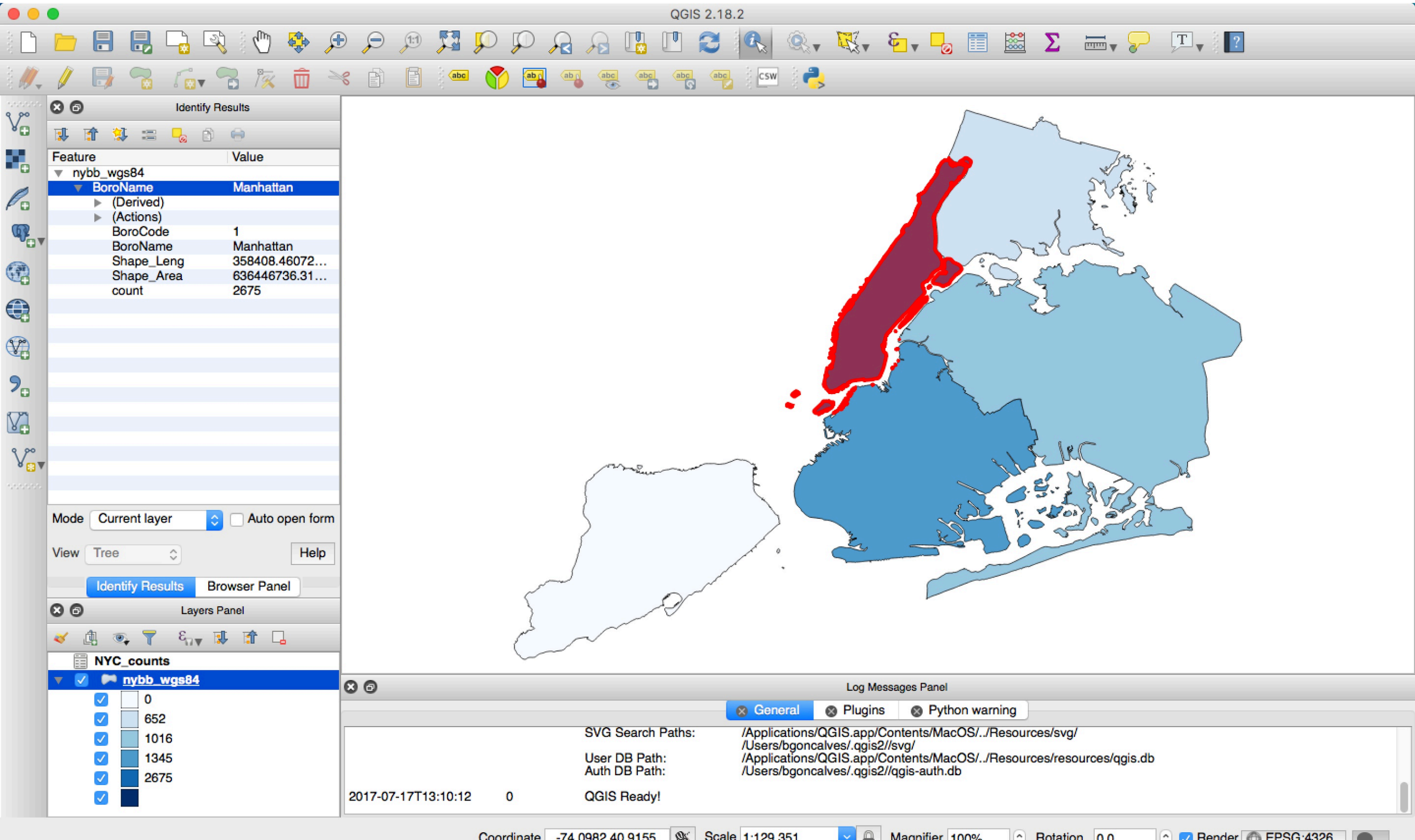
shapefile_aggregate.py

QGIS Demo

Choropleth

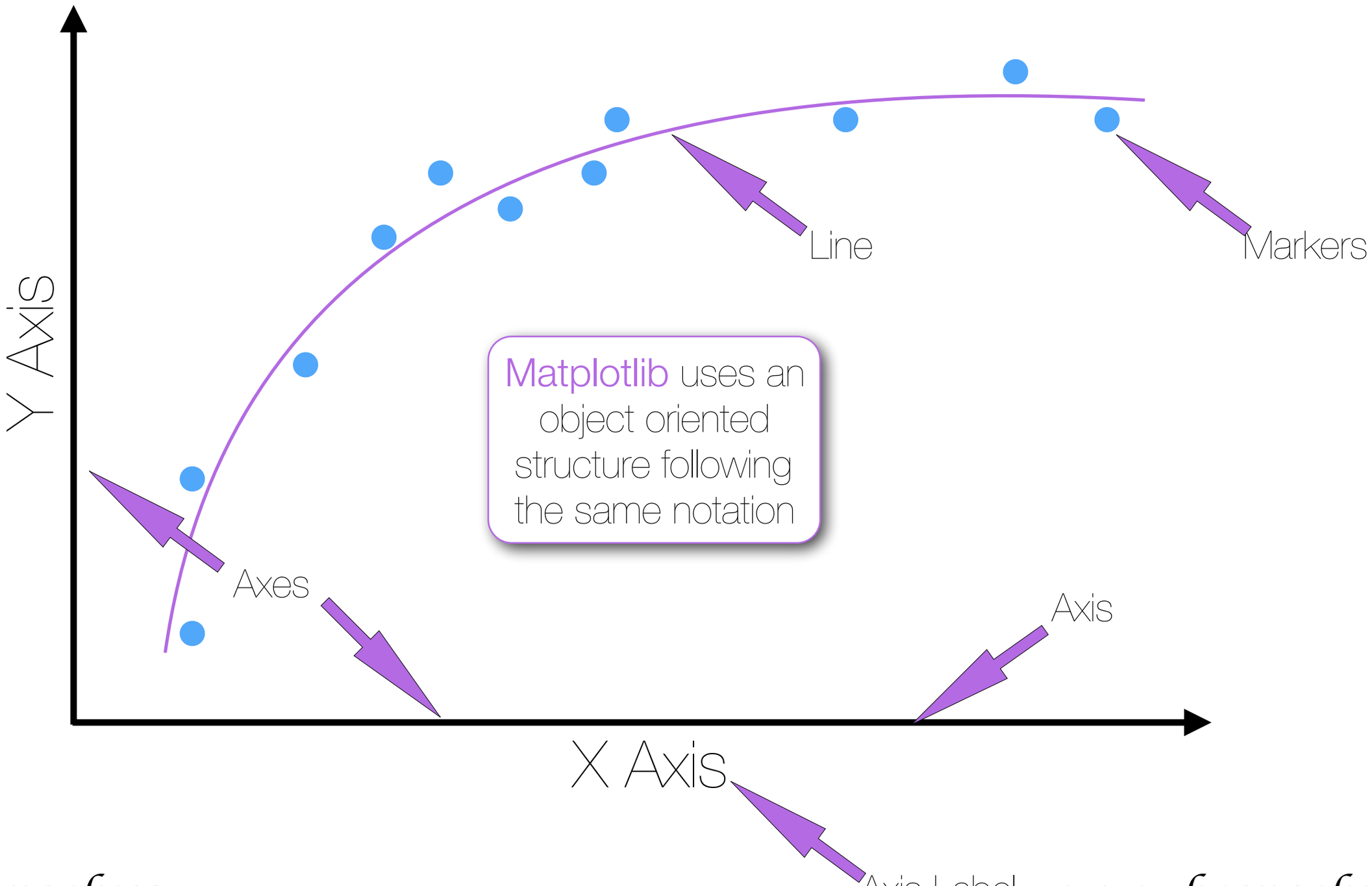

QGIS 2.18.2
Identify Results
Feature
Value
Mode
Current layer
Auto open form
View
Tree
Help
Identify Results
Browser Panel
Layers Panel
NYC_counts
nybb_wgs84
0
652
1016
1345
2675
Log Messages Panel
General
Plugins
Python warning
SVG Search Paths: /Applications/QGIS.app/Contents/MacOS/../Resources/svg/
/Users/bgoncalves/.qgis2//svg/
User DB Path: /Applications/QGIS.app/Contents/MacOS/../Resources/resources/qgis.db
Auth DB Path: /Users/bgoncalves/.qgis2//qgis-auth.db
2017-07-17T13:10:12 0 QGIS Ready!
Coordinate -74.2670,40.8112
Scale 1:129,351
Magnifier 100%
Rotation 0.0
Render
EPSG:4326

Choropleth

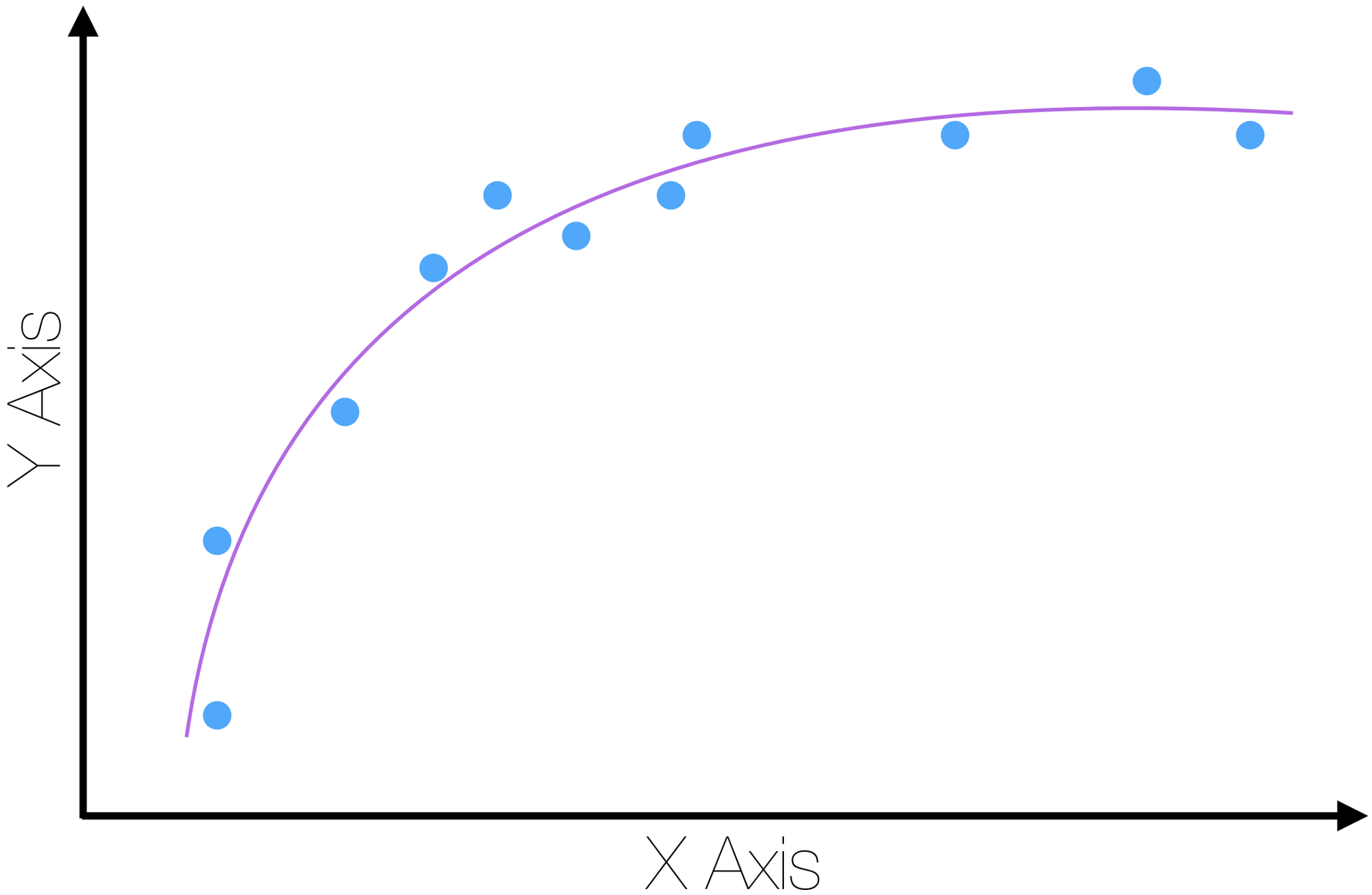

QGIS 2.18.2
Identify Results
Feature
Value
nybb_wgs84
BoroName
Manhattan
(Derived)
(Actions)
BoroCode
1
BoroName
Manhattan
Shape_Leng
358408.46072…
Shape_Area
636446736.31…
count
2675
Mode
Current layer
Auto open form
View
Tree
Help
Identify Results
Browser Panel
Layers Panel
NYC_counts
nybb_wgs84
0
652
1016
1345
2675
Log Messages Panel
General
Plugins
Python warning
SVG Search Paths:
/Applications/QGIS.app/Contents/MacOS/../Resources/svg/
/Users/bgoncalves/.qgis2//svg/
User DB Path:
/Applications/QGIS.app/Contents/MacOS/../Resources/resources/qgis.db
Auth DB Path:
/Users/bgoncalves/.qgis2//qgis-auth.db
2017-07-17T13:10:12
0
QGIS Ready!
Coordinate
-74.0982,40.9155
Scale
1:129,351
Magnifier
100%
Rotation
0.0
Render
EPSG:4326

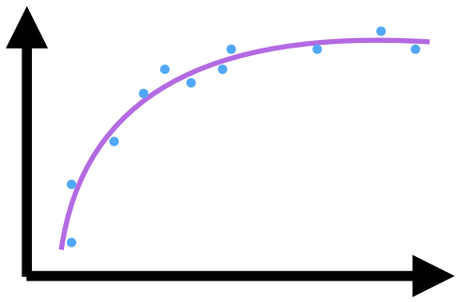
Matplotlib

Basic Plotting



Basic Plotting

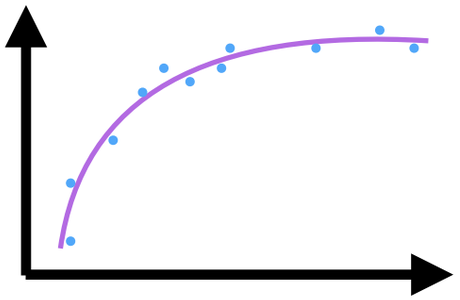


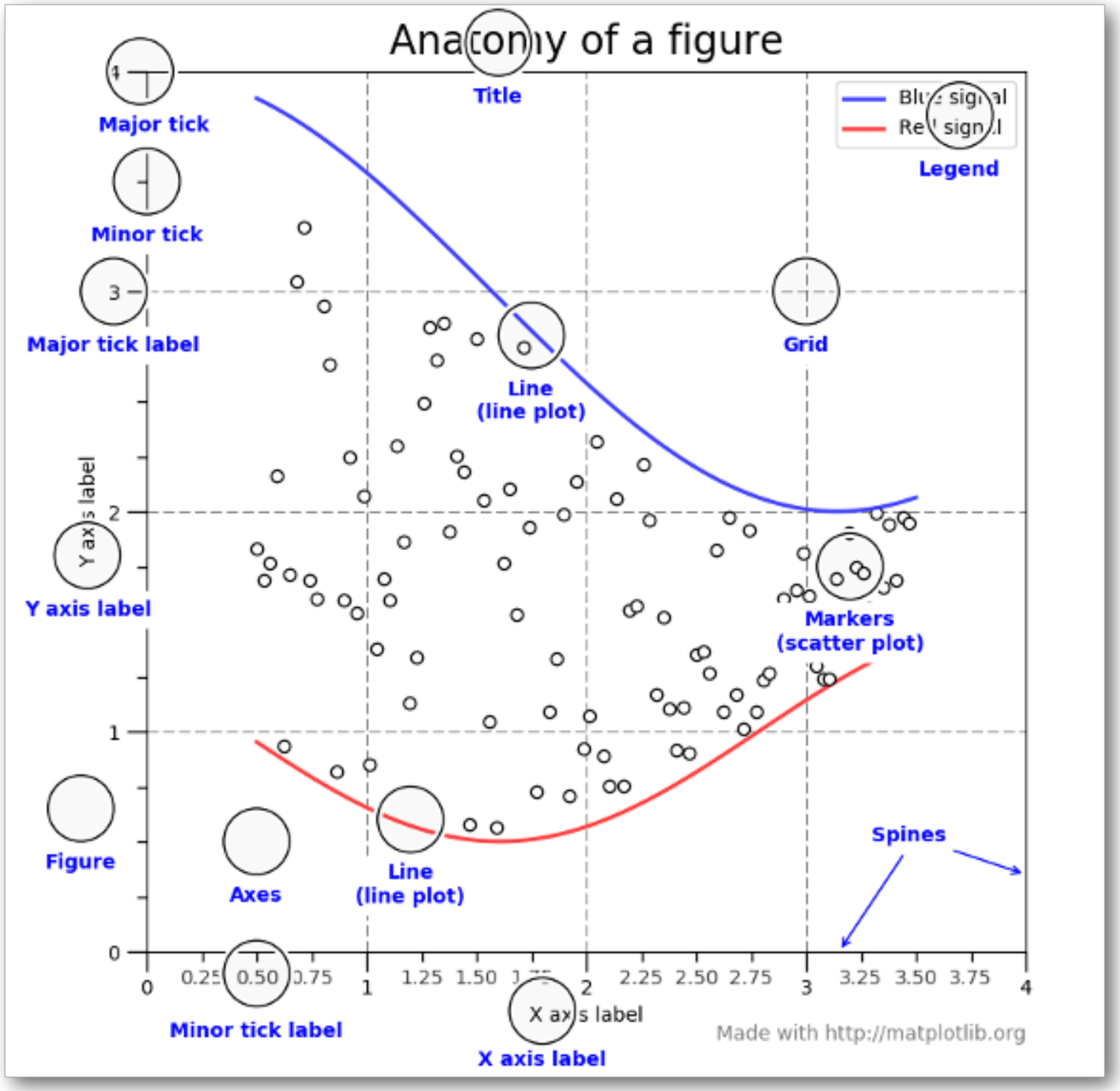


Basic Plotting

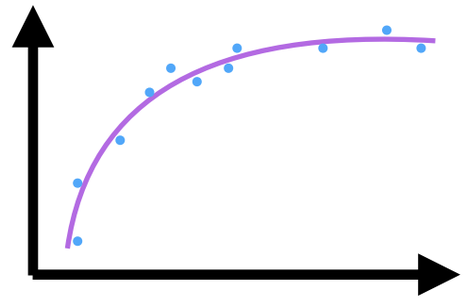
- Matplotlib uses an object oriented structure following an intuitive notation
- Each Axes object contains one or more Axis objects.
- A Figure is a set of one or more Axes.
- Each Axes is associated with exactly one Figure and each set of Markers is associated with exactly one Axes.
- In other words, Markers/Lines represent a dataset that is plotted against one or more Axis. An Axes object is (effectively) a subplot of a Figure.

Basic Plotting - Programmatically!





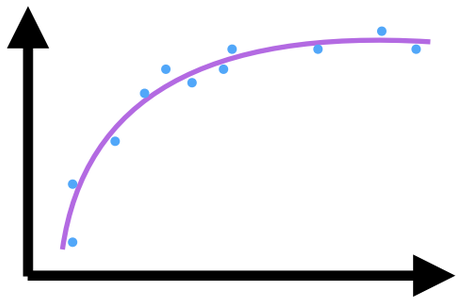
Basic Plotting - Programmatically!



https://matplotlib.org/2.0.0/

- While the Figure object (plt) controls the way in which the figure is displayed.
  - .gca() - Get the current Axes, creating one if necessary
  - .show() - Show the final figure
  - .savefig("filename.ext") - Save the figure to "filename.ext" where ".ext" defines the format the saved image ()

```
filetypes = {'ps': 'Postscript', 'eps': 'Encapsulated Postscript', 'pdf': 'Portable Document Format',
'pgf': 'PGF code for LaTeX', 'png': 'Portable Network Graphics', 'raw': 'Raw RGBA bitmap', 'rgba': 'Raw
RGBA bitmap', 'svg': 'Scalable Vector Graphics', 'svgz': 'Scalable Vector Graphics', 'jpg': 'Joint
Photographic Experts Group', 'jpeg': 'Joint Photographic Experts Group', 'tif': 'Tagged Image File
Format', 'tiff': 'Tagged Image File Format'}
```



Basic Plotting - Programmatically!

- The first step is to import the pyplot module from matplotlib and instanciating a Figure object:

```
import matplotlib.pyplot as plt
fig = plt.figure()
```

- The convention is to import pyplot as plt
- To create subplots (Axes) you use .subplots(nrows, ncols, sharex=False, sharey=False) instead of .figure(). set sharex and/or sharey to True to keep the same scale in both cases.
- .subplots - returns a (fig, ax_lst) tuple where ax_lst is a list of Axes and fig is the Figure.
- Axes have several methods of interest:
  - .plot(x, y) - Make a scatter or line plot from a list of x, y coordinates.
  - .imshow(mat) - Plot a matrix as if it were an image. Element 0,0 is plotted in the top right corner.
  - .bar(x, y) - Make a bar plot where x is a list of the lower left coordinates of each bar and y is the respective height.
  - .pie(values, labels=labels) - Produce a pie plot out of a list of values list and labeled with labels
  - .savefig(filename) - Write the current figure as an static image

Challenge - Matplotlib

- Plot the function:

$$f(x) = x^2$$

- and

$$g(x) = (x-3)^3$$

- side by side (separate Axes objects) for:

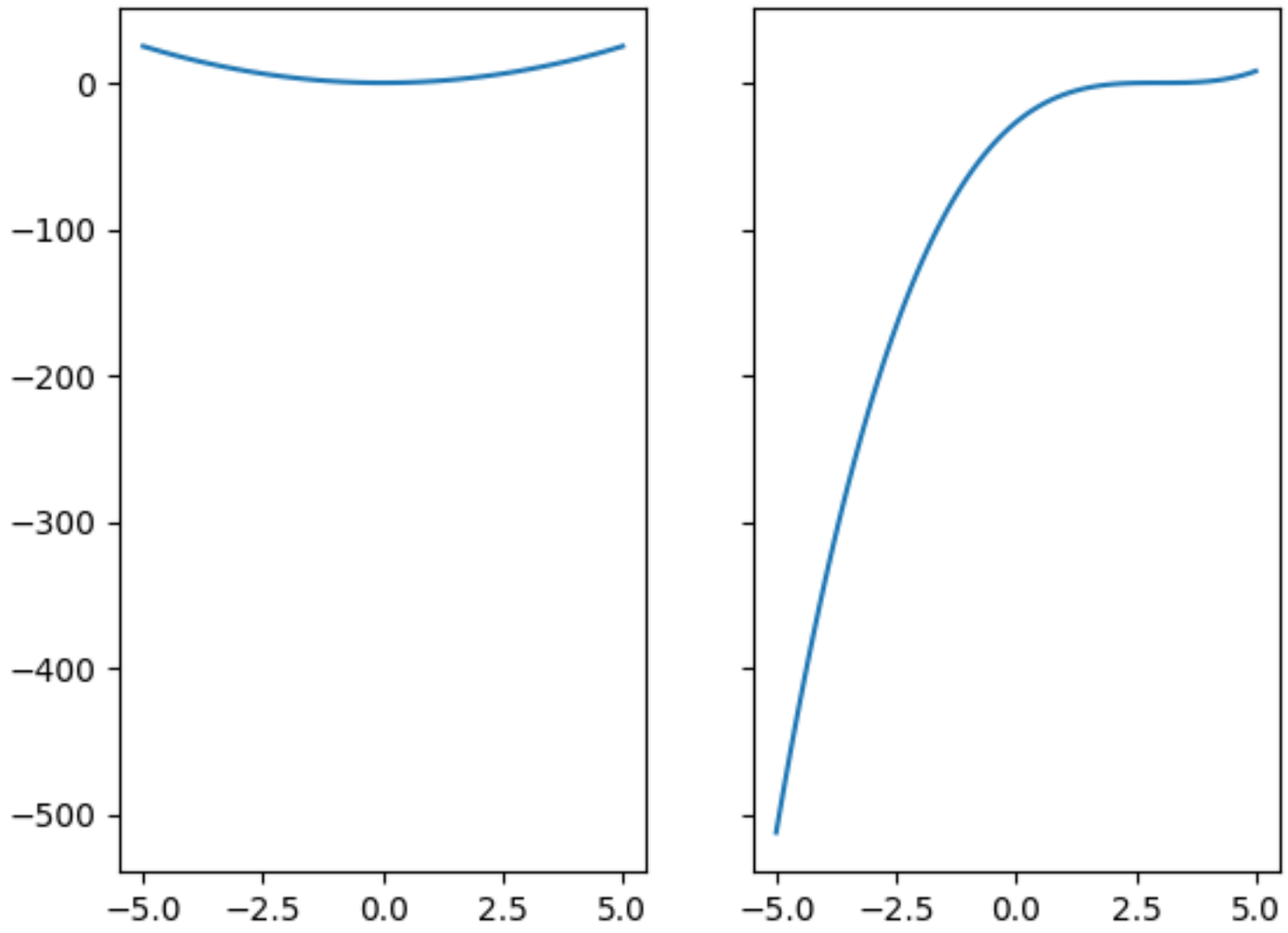
$$x \in [-5, 5]$$

Challenge - Matplotlib

```python
import matplotlib.pyplot as plt
import numpy as np

fig, ax_lst = plt.subplots(1, 2, sharex=True, sharey=True)

x = np.linspace(-5, 5, 100)
y1 = np.power(x, 2.)
y2 = np.power(x-3, 3.)

ax_lst[0].plot(x, y1)
ax_lst[1].plot(x, y2)
fig.savefig('demo.png')
```

matplotlib_demo.py

Challenge - Matplotlib



Challenge - Matplotlib

- Plot the function:

$$f\left(x\right) = x^2$$

- and

$$g\left(x\right) = \left(x - 3\right)^3$$

- side by side (separate Axes objects) for:

$$x = 1, 2, 3, 4, 5$$

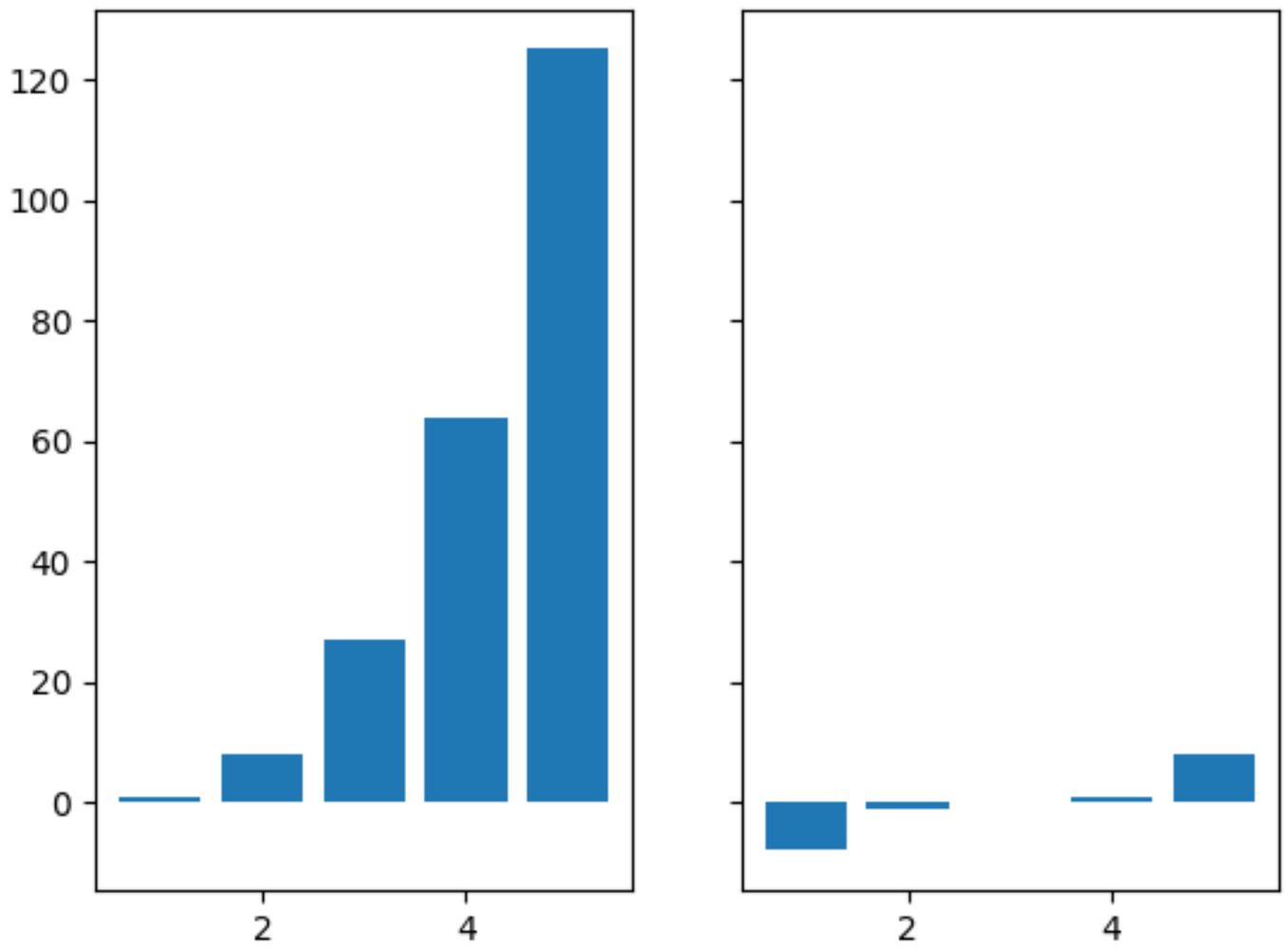
- As a bar chart

Challenge - Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np

fig, ax_lst = plt.subplots(1, 2, sharex=True,
sharey=True)

x = np.arange(1, 6)
y1 = np.power(x, 3.)
y2 = np.power(x-3, 3.)

ax_lst[0].bar(x, y1)
ax_lst[1].bar(x, y2)
fig.savefig('demo_bar.png')
```

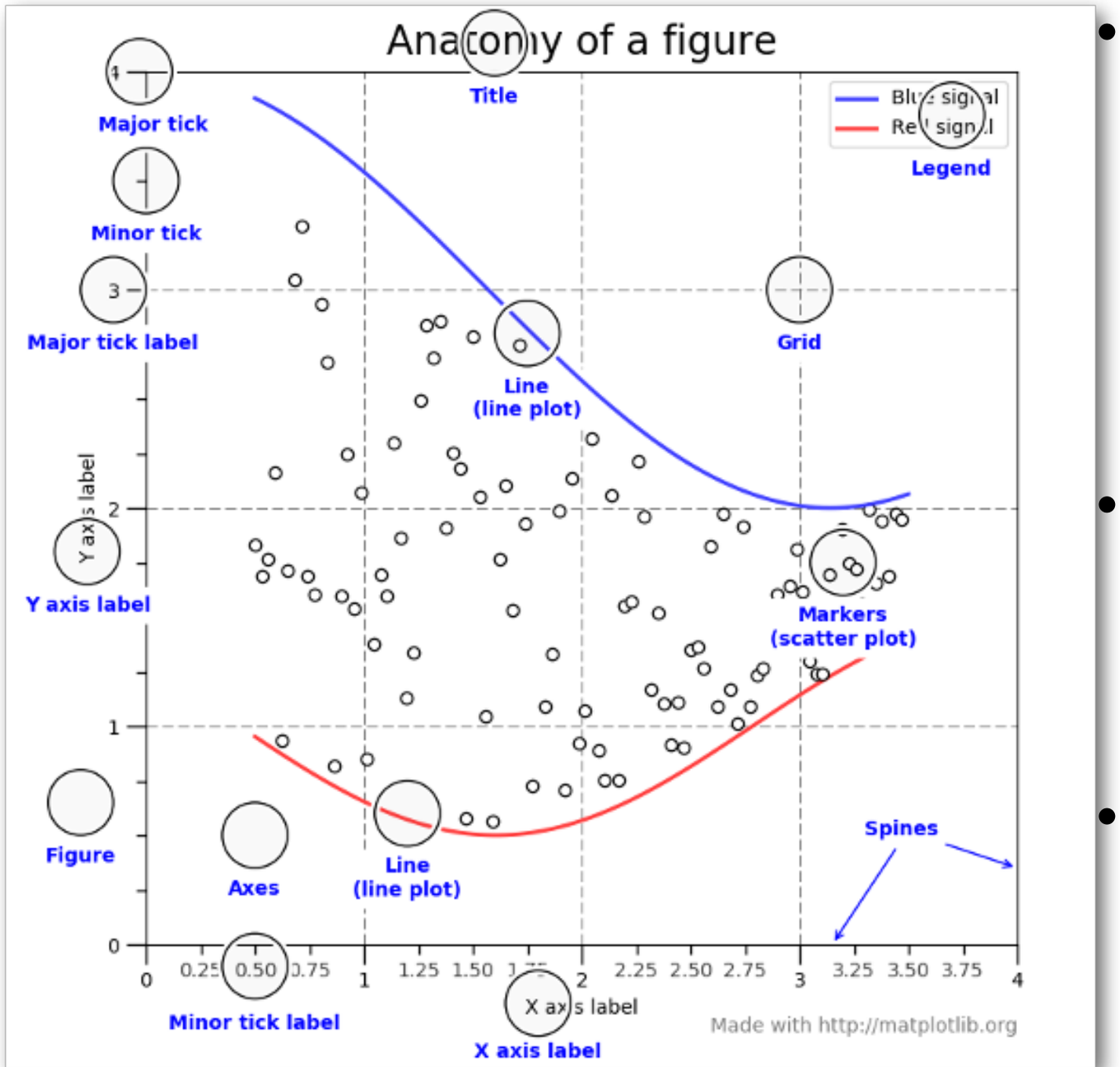
matplotlib_demo_bar.py

Challenge - Matplotlib



Matplotlib - decorations

https://matplotlib.org/2.0.0/



- The respective functions are named in an intuitive wa
  Every Axes object has as methods:
  - .set_xlabel(label)
  - .set_ylabel(label)
  - .set_title(title)
- And axis limits can be set using:
  - .set_xlim(xmin, xmax)
  - .set_ylim(ymin, ymax)
- Tick marks and labels are set using:
  - .set_xticks(ticks)/.set_yticks(ticks)
  - .set_xticklabels(labels)/.set_yticklabels(labels)

Matplotlib - Images

https://matplotlib.org/2.0.0/

- .imshow(fig) - Display an image on a set of axes.

- fig can be any matrix of numbers.

- Further plotting can occur by simply using the functions described above

Challenge - imshow

- Plot the population distribution of the US (similarly to last time) from

geofiles/US_pop.asc

- Add a line connecting two points:

(39.163355, -86.523435)

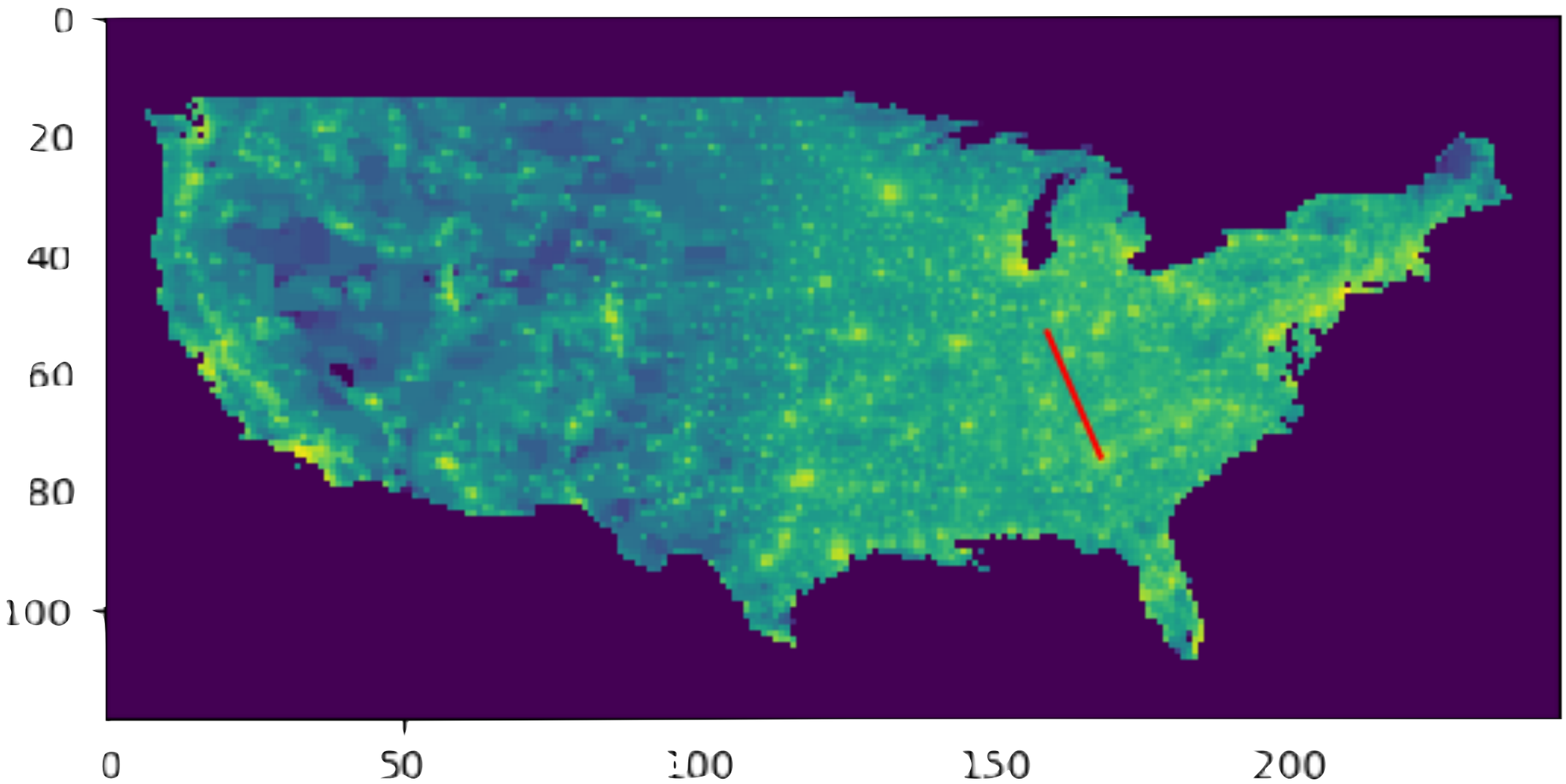
(33.761926, -84.404820)

Challenge - imshow

```
import numpy as np
import matplotlib.pyplot as plt

(…)

fig, ax = plt.subplots(1,1)
data, xllcorner, yllcorner, cellsize  = load_asc('../Lecture V/geofiles/US_pop.asc')
ax.imshow(np.log(data+1))

x1, y1 = map_points(xllcorner, yllcorner, cellsize, data.shape[0], -86.523435, 39.163355)
x2, y2 = map_points(xllcorner, yllcorner, cellsize, data.shape[0], -84.404820, 33.761926)

ax.plot([x1, x2], [y1, y2], 'r-')
fig.savefig('Manhattan_ASC.png')
```


matplotlib_imshow.py

Challenge - imshow



Challenge - Overlap a raster and a shapefile

https://www.census.gov/geo/maps-data/data/cbf/cbf_state.html

- Replot the US population file:

geofiles/US_pop.asc

- And overlap the shape file

geofiles/48States/48States.shp

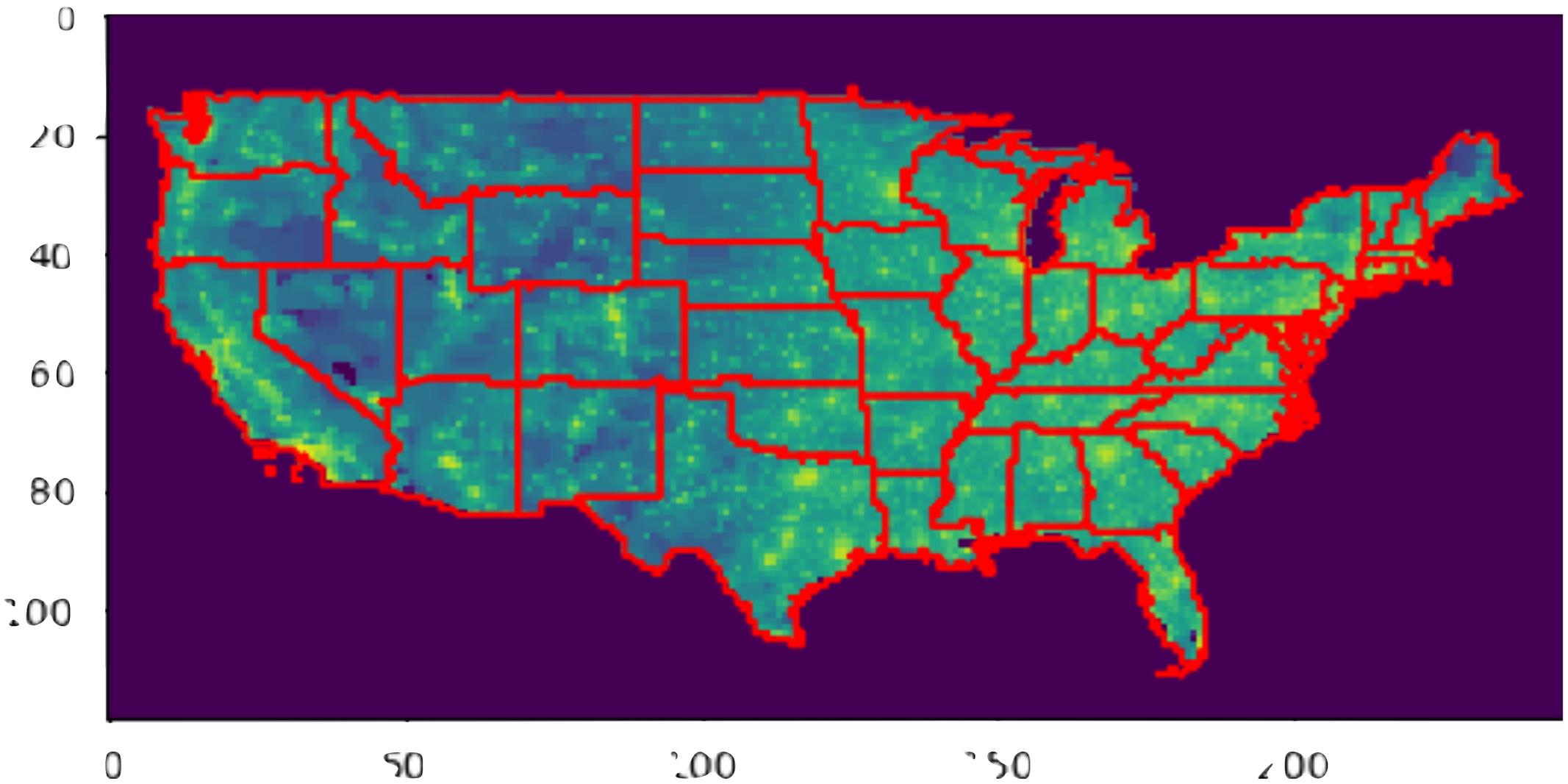
- On top of it. Don't forget that you have to convert the lat and lon values to matrix cell coordinates.

Challenge - Overlap a raster and a shapefile

```
import numpy as np
import matplotlib.pyplot as plt
import shapefile

(…)

fig, ax = plt.subplots(1,1)
data, xllcorner, yllcorner, cellsize  = load_asc('../Lecture V/geofiles/US_pop.asc')
ax.imshow(np.log(data+1))

shp = shapefile.Reader('geofiles/48States/48States.shp')

pos = None
count = 0
for shape in shp.iterShapes():
        points = np.array(shape.points)
        parts = shape.parts
        parts.append(len(shape.points))

        for i in range(len(parts)-1):
            positions = []

            for j in range(parts[i+1]-parts[i]):
                x_orig = points.T[0][parts[i]+j]
                y_orig = points.T[1][parts[i]+j]
                x, y = map_points(xllcorner, yllcorner, cellsize, data.shape[0], x_orig, y_orig)
                positions.append([x, y])

            positions = np.array(positions)

            ax.plot(positions.T[0], positions.T[1], 'r-')


fig.savefig('US overlap.png')
```

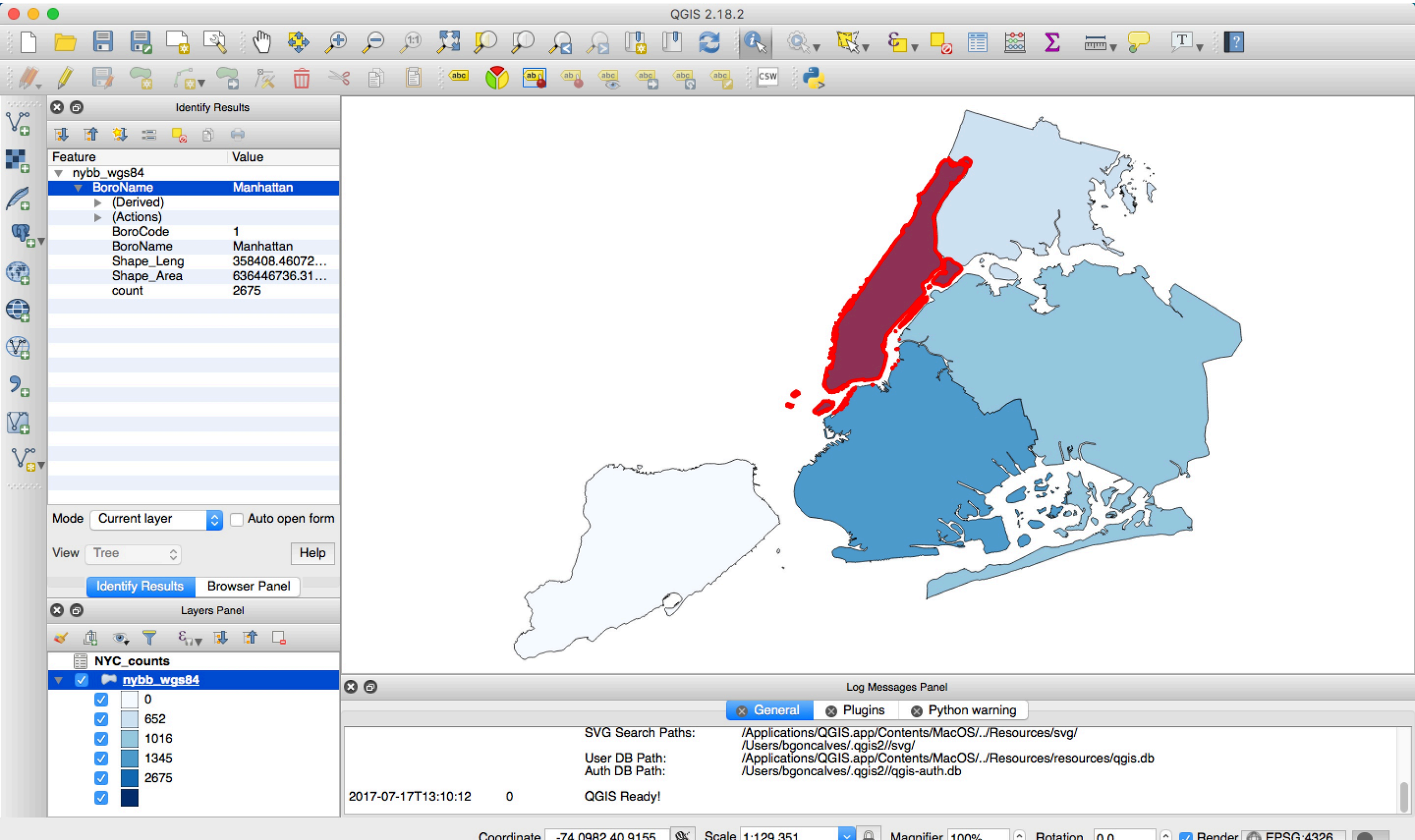
matplotlib_overlap.py

Challenge - Overlap a raster and a shapefile



@bgoncalves

Choropleth


QGIS 2.18.2
Identify Results
Feature
Value
nybb_wgs84
BoroName
Manhattan
(Derived)
(Actions)
BoroCode
1
BoroName
Manhattan
Shape_Leng
358408.46072…
Shape_Area
636446736.31…
count
2675
Mode
Current layer
Auto open form
View
Tree
Help
Identify Results
Browser Panel
Layers Panel
NYC_counts
nybb_wgs84
0
652
1016
1345
2675
Log Messages Panel
General
Plugins
Python warning
SVG Search Paths:
/Applications/QGIS.app/Contents/MacOS/../Resources/svg/
/Users/bgoncalves/.qgis2//svg/
User DB Path:
/Applications/QGIS.app/Contents/MacOS/../Resources/resources/qgis.db
Auth DB Path:
/Users/bgoncalves/.qgis2//qgis-auth.db
2017-07-17T13:10:12
0
QGIS Ready!
Coordinate
-74.0982,40.9155
Scale
1:129,351
Magnifier
100%
Rotation
0.0
Render
EPSG:4326

Matplotlib patches

https://matplotlib.org/2.0.0/api/patches_api.html

- matplotlib.patches are essentially colored areas:
    - Wedge(xy, r, theta1, theta2, width)
    - Circle(xy, radius)
    - Rectangle(xy, width, height, angle)
    - Polygon(coords, closed=True)
- xy is a tuple representing the center (Circle, Wedge), or lower left corner (Rectangle)
- coords is a Nx2 array of the coordinates of each point

Matplotlib patches

https://matplotlib.org/2.0.0/api/patches_api.html

- matplotlib.patches can be grouped in to matplotlib.collections.PatchCollections for ease of handling and for faster plotting

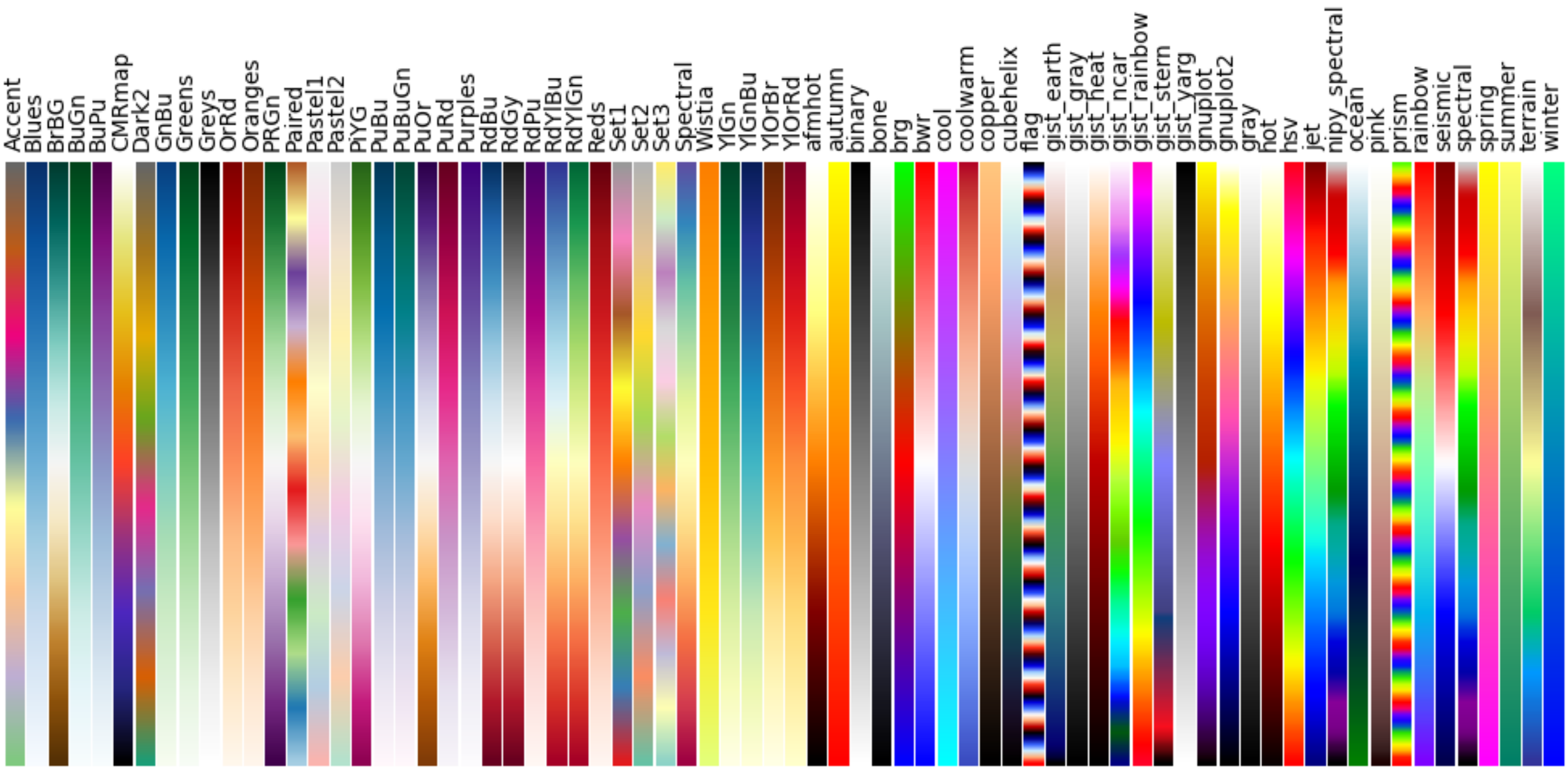
- PatchCollection(patches, facecolor, edgecolor='k') where patches and facecolor are lists of patches and colors.

- PatchCollection's can be added to a given Axis object with .add_collection()

Matplotlib color maps

https://matplotlib.org/users/colormaps.html

- When plotting we often want to define a continuous color scale.
- In matplotlib these are called cmap (for colormap) and live in matplotlib.pyplot.cm



matplotlib color maps

https://matplotlib.org/users/colormaps.html

- colormaps can be selected by using .get_cmap("name") that returns a Colormap instance
- Colormap(x) where x is a value between 0 and 1
- arbitrary values can be easily converted to the [0,1] range using a matplotlib.colors.Normalize object
- Normalize(vmin, vmax) - Creates a Normalize object that maps from [vmin, vmax] to [0, 1]
- "Colormapping typically involves two steps: a data array is first mapped onto the range 0-1 using an instance of Normalize or of a subclass; then this number in the 0-1 range is mapped to a color using an instance of a subclass of Colormap"

We can now make arbitrary choropleths using patches and colormaps!

Choropleth

```
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
from matplotlib.collections import PatchCollection
from matplotlib import patches
import shapefile
import numpy as np

shp = shapefile.Reader('geofiles/nybb_15cc/nybb_wgs84.shp')
boros = {}

for shaperecord in shp.iterShapeRecords():
    boro_id = int(shaperecord.record[0])
    parts = list(shaperecord.shape.parts)
    parts.append(len(shaperecord.shape.points))
    boros[boro_id] = []

    for i in range(0, len(parts)-1):
        boros[boro_id].append(patches.Polygon(shaperecord.shape.points[parts[i]:parts[i+1]-1]))
```


plot_cloropleth.py

Choropleth

```
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
from matplotlib.collections import PatchCollection
from matplotlib import patches
import shapefile
import numpy as np

shp = shapefile.Reader('geofiles/nybb_15cc/nybb_wgs84.shp')
boros = {}

for shaperecord in shp.iterShapeRecords():
    boro_id = int(shaperecord.record[0])
    parts = list(shaperecord.shape.parts)
    parts.append(len(shaperecord.shape.points))
    boros[boro_id] = []

    for i in range(0, len(parts)-1):
        boros[boro_id].append(patches.Polygon(shaperecord.shape.points[parts[i]:parts[i+1]-1]))
```

@bgoncalves

plot_cloropleth.py

```
data = {}
line_count = 0

for line in open("NYC_counts.csv"):
    fields = line.strip().split(',')

    line_count += 1

    if line_count == 1:
        header = dict(zip(fields, range(len(fields))))
        continue

    data[int(fields[header["id"]])] = int(fields[header["count"]])

Min = np.min(list(data.values()))
Max = np.max(list(data.values()))
norm = Normalize(vmin=Min, vmax=Max)
cmap = plt.get_cmap('Blues')

patches = []
colors = []

for boro_id in boros.keys():
    patches.extend(boros[boro_id])

    if boro_id in boros:
        colors.extend([cmap(norm(data[boro_id])) for i in range(len(boros[boro_id]))])
    else:
        colors.append([cmap(norm(Min)) for i in range(len(boros[boro_id]))])

fig = plt.figure()
ax = plt.gca()

bbox = shp.bbox
ax.set_xlim(bbox[0], bbox[2])
ax.set_ylim(bbox[1], bbox[3])
ax.add_collection(PatchCollection(patches, facecolor=colors, edgecolor='k'))

fig.savefig('NYC_counts.png')
plt.close(fig)
```

plot_choropleth.py

```
data = {}
line_count = 0

for line in open("NYC_counts.csv"):
    fields = line.strip().split(',')

    line_count += 1

    if line_count == 1:
        header = dict(zip(fields, range(len(fields))))
        continue

    data[int(fields[header["id"]])] = int(fields[header["count"]])

Min = np.min(list(data.values()))
Max = np.max(list(data.values()))
norm = Normalize(vmin=Min, vmax=Max)
cmap = plt.get_cmap('Blues')

patches = []
colors = []

for boro_id in boros.keys():
    patches.extend(boros[boro_id])

    if boro_id in boros:
        colors.extend([cmap(norm(data[boro_id])) for i in range(len(boros[boro_id]))])
    else:
        colors.append([cmap(norm(Min)) for i in range(len(boros[boro_id]))])

fig = plt.figure()
ax = plt.gca()

bbox = shp.bbox
ax.set_xlim(bbox[0], bbox[2])
ax.set_ylim(bbox[1], bbox[3])
ax.add_collection(PatchCollection(patches, facecolor=colors, edgecolor='k'))

fig.savefig('NYC_counts.png')
plt.close(fig)
```

plot_choropleth.py

Matplotlib Basemap

Basemap

https://matplotlib.org/basemap/

- The Basemap module is the workhorse and returns a Basemap object when instantiated.
- The Basemap object has many useful methods to assist in drawing a map:
  - .drawcoastlines() - To draw the coastlines of continents
  - .drawmapboundary() - To draw the boundary of the map
  - .fillcontinents() - add color to the continents
  - etc…
- The constructor for Basemap can take many different arguments to be able to handle different projections, but it defaults to the Plate Carrée projection centered at (0, 0)
- The minimal map is simply:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

map = Basemap()
map.drawcoastlines()
plt.savefig('basemap_demo.png')
```

- Please note that with the .drawcoastlines() call nothing is plotted as our map has no content.

Basemap

https://matplotlib.org/basemap/

- We can also visualize just specific regions by setting the bbox and center coordinates by setting

llcrnrlon, llcrnrlat, urcrnrlon, urcrnrlat

- And

lat_0, lon_0

- Respectively.
- We can convert arbitrary lat, lon values to map coordinates by calling the map() object directly.
- After we obtain the map coordinates we can add them to the map by calling the .plot(x, y) method of the map object.

Basemap Example

https://matplotlib.org/basemap/

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

map = Basemap(projection='ortho', lat_0=0, lon_0=0)

map.drawmapboundary(fill_color='aqua')
map.fillcontinents(color='coral',lake_color='aqua')
map.drawcoastlines()

x, y = map(0, 0)

map.plot(x, y, marker='D',color='m')

plt.savefig('globe.png')
```

basemap_example.py

Basemap Example