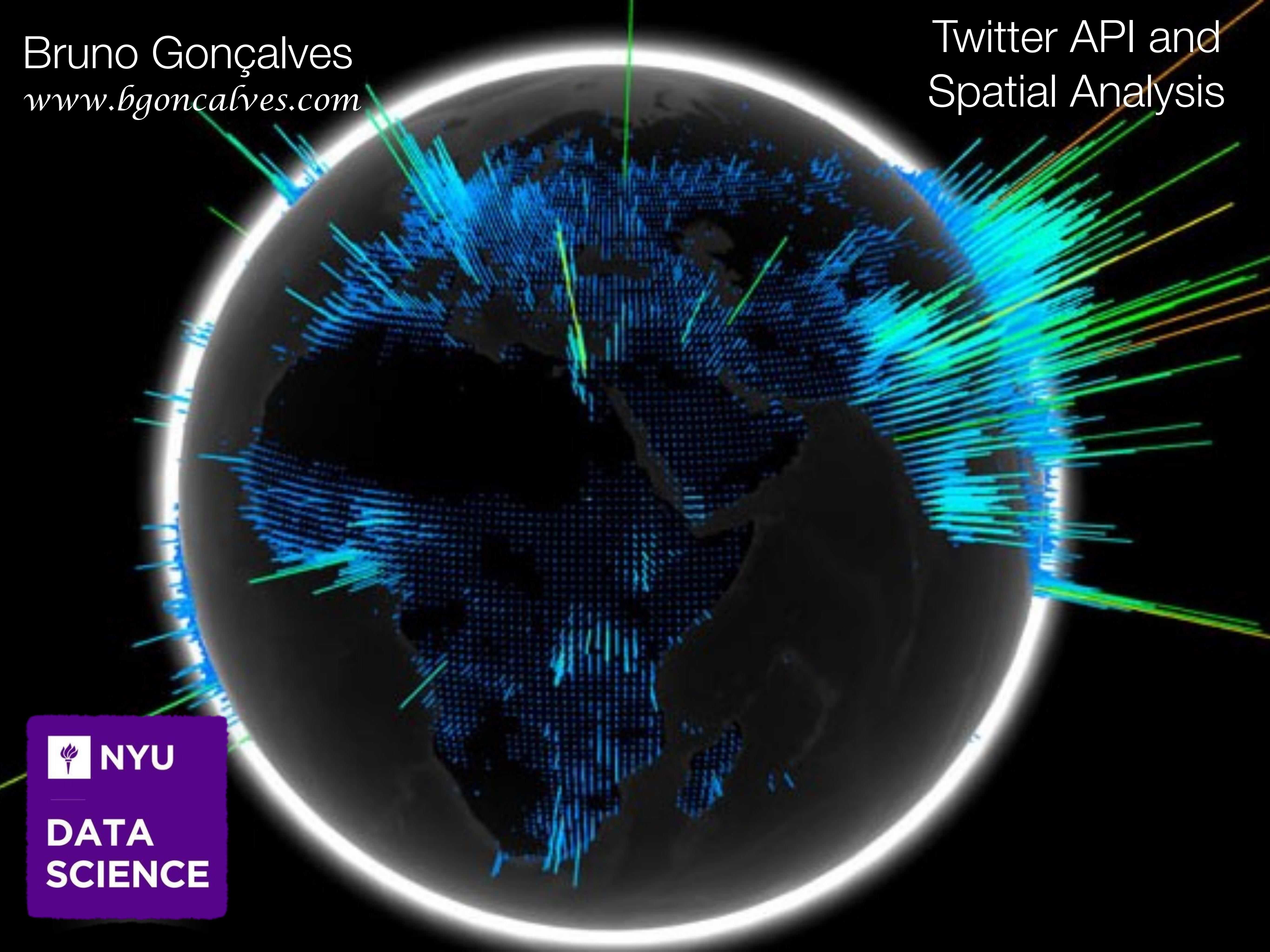
Bruno Gonçalves

www.bgoncalves.com

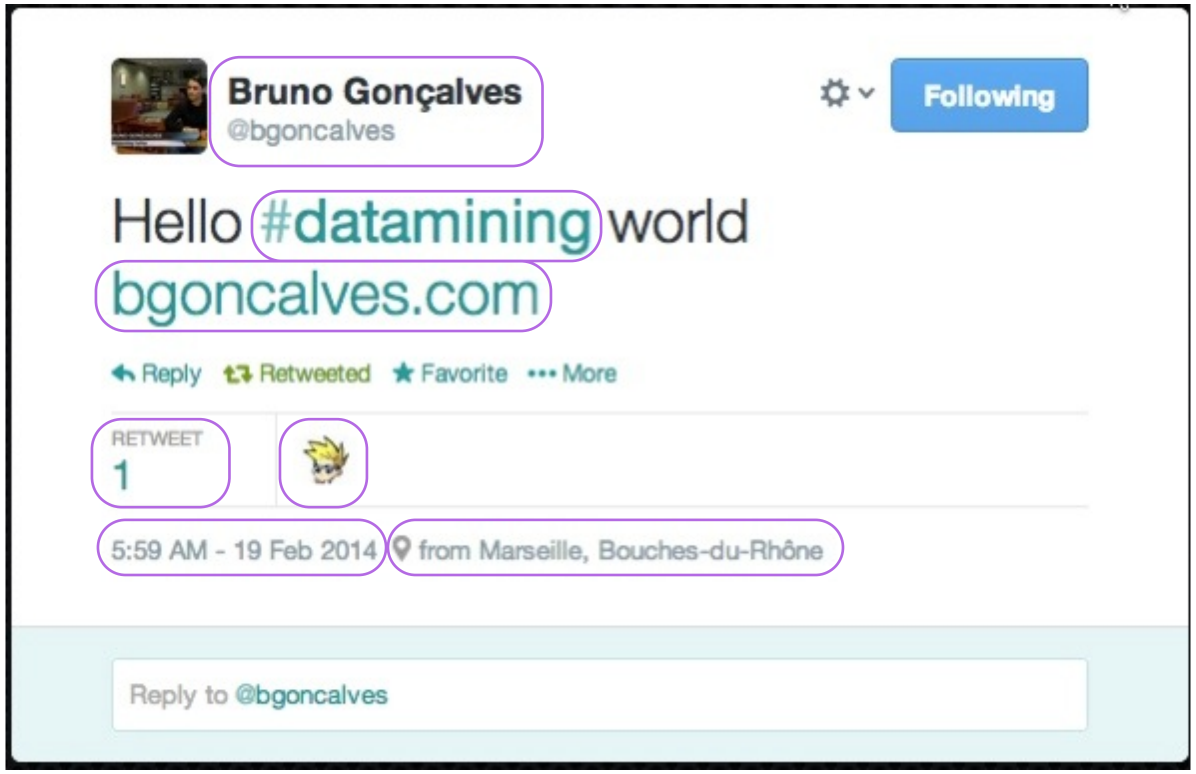
Twitter API and Spatial Analysis



NYU

DATA SCIENCE

Anatomy of a Tweet



Bruno Gonçalves
@bgoncalves

Hello #datamining world
bgoncalves.com

RETWEET
1

5:59 AM - 19 Feb 2014
from Marseille, Bouches-du-Rhône

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

```
[u'follow_request_sent',
 u'profile_use_background_image',
 u'default_profile_image',
 u'id',
 u'profile_background_image_url_https',
 u'verified',
 u'profile_text_color',
 u'profile_image_url_https',
 u'profile_sidebar_fill_color',
 u'entities',
 u'followers_count',
 u'profile_sidebar_border_color',
 u'id_str',
 u'profile_background_color',
 u'listed_count',
 u'is_translation_enabled',
 u'utc_offset',
 u'statuses_count',
 u'description',
 u'friends_count',
 u'location',
 u'profile_link_color',
 u'profile_image_url',
 u'following',
 u'geo_enabled',
 u'profile_banner_url',
 u'profile_background_image_url',
 u'screen_name',
 u'lang',
 u'profile_background_tile',
 u'favourites_count',
 u'name',
 u'notifications',
 u'url',
 u'created_at',
 u'contributors_enabled',
 u'time_zone',
 u'protected',
 u'default_profile',
 u'is_translator']
```

Anatomy of a Tweet

```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

```
u"I'm at Terminal Rodovi\xe1rio de Feira de Santana
 (Feira de Santana, BA) http://t.co/WirvdHwYMq"
```

```
u'<a href="http://foursquare.com" rel="“nofollow">
  foursquare</a>'
```

```
[u'symbols',
 u'user_mentions',
 u'hashtags',
 u'urls']
```

```
[u'type',
 u'coordinates']
```

Anatomy of a Tweet

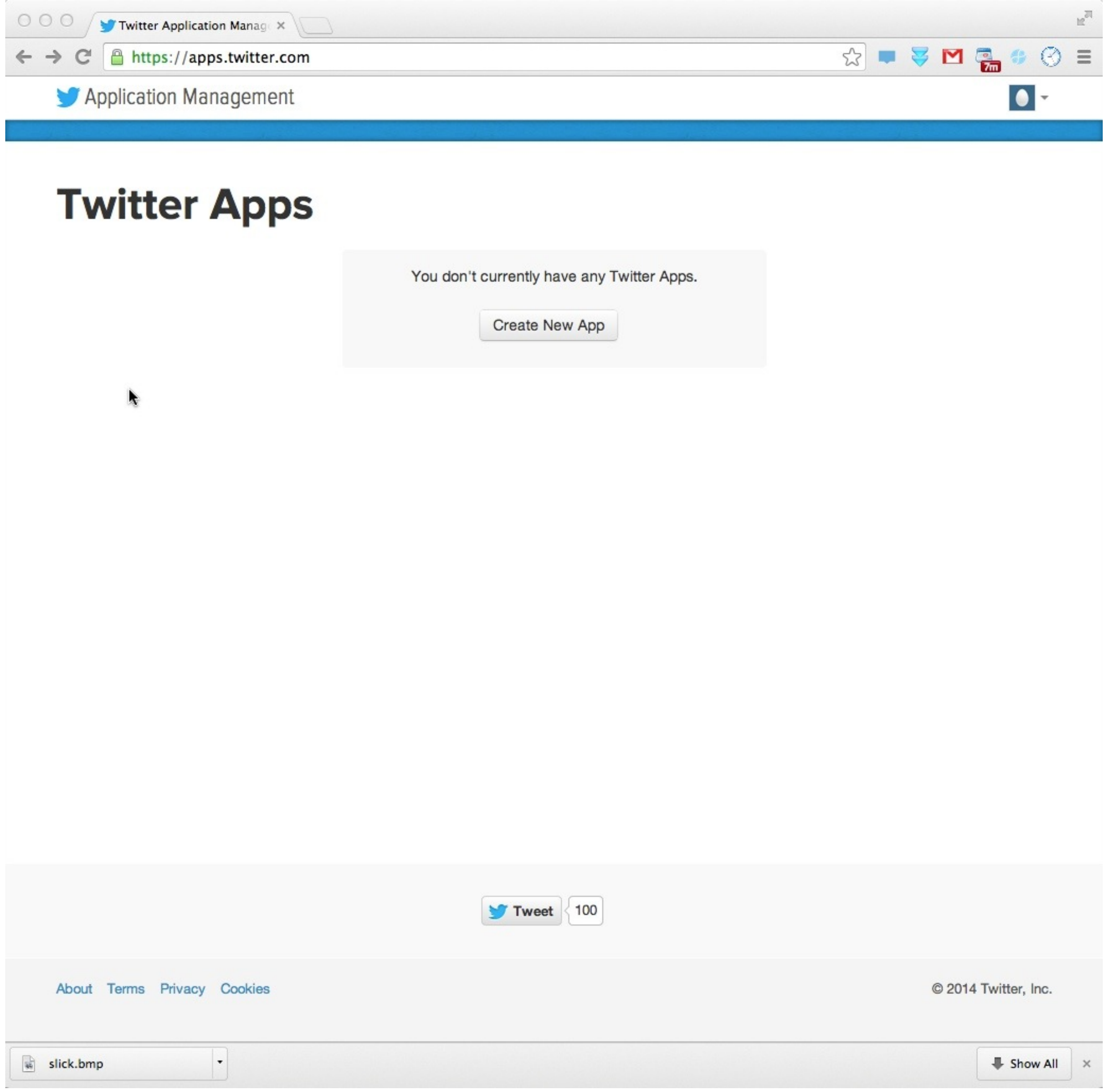
```
[u'contributors',
 u'truncated',
 u'text',
 u'in_reply_to_status_id',
 u'id',
 u'favorite_count',
 u'source',
 u'retweeted',
 u'coordinates',
 u'entities',
 u'in_reply_to_screen_name',
 u'in_reply_to_user_id',
 u'retweet_count',
 u'id_str',
 u'favorited',
 u'user',
 u'geo',
 u'in_reply_to_user_id_str',
 u'possibly_sensitive',
 u'lang',
 u'created_at',
 u'in_reply_to_status_id_str',
 u'place',
 u'metadata']
```

```
u"I'm at Terminal Rodovi\xe1rio de Feira de Santana
 (Feira de Santana, BA) http://t.co/WirvdHwYMq"
```

```
u'<a href="http://foursquare.com" rel=“nofollow">
  foursquare</a>'
```

```
[u'symbols',
 u'user_mentions',
 u'hashtags',{u'display_url': u'4sq.com/1k5MeYF',
 u'urls']      u'expanded_url': u'http://4sq.com/1k5MeYF',
               u'indices': [70, 92],
[u'type',      u'url': u'http://t.co/WirvdHwYMq'}
 u'coordinates']
```

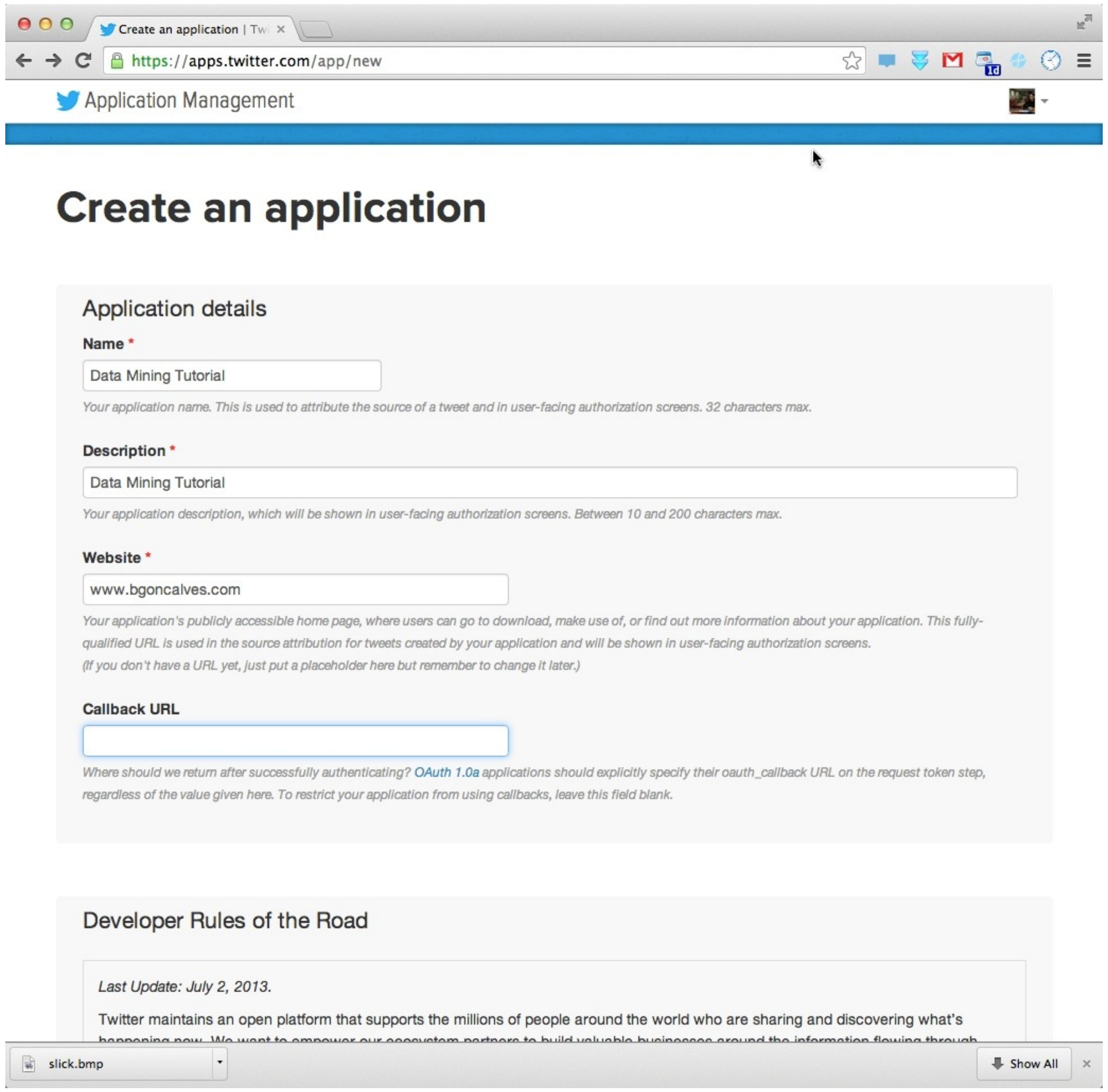
Registering an Application



Twitter Apps

You don't currently have any Twitter Apps.

Registering an Application



Create an application

Application details

Name *

Data Mining Tutorial

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Data Mining Tutorial

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

www.bgoncalves.com

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

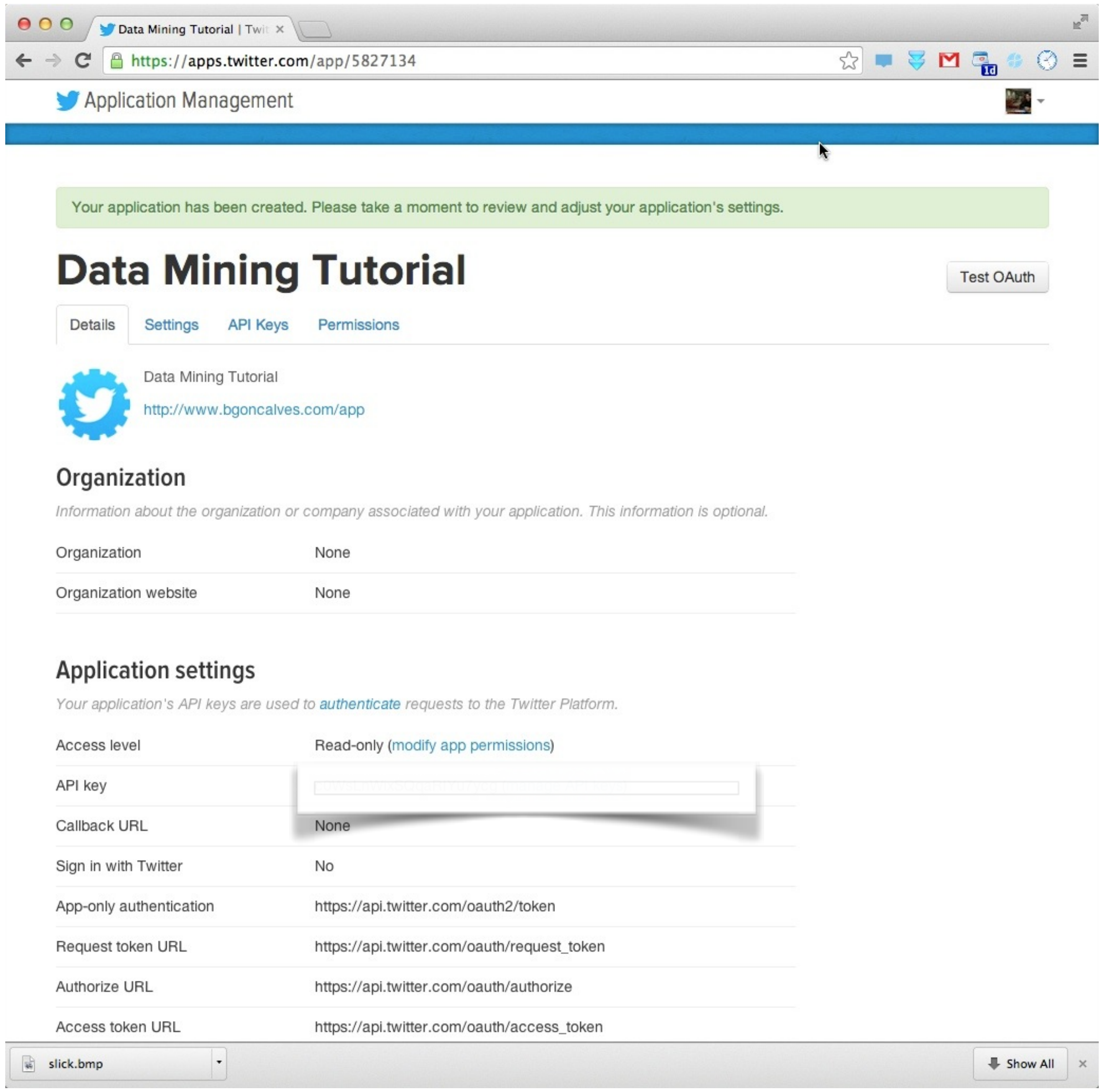
Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Developer Rules of the Road

Last Update: July 2, 2013.

Twitter maintains an open platform that supports the millions of people around the world who are sharing and discovering what's happening now. We want to empower our ecosystem partners to build valuable businesses around the information flowing through

Registering an Application


Data Mining Tutorial | Twi
https://apps.twitter.com/app/5827134
Application Management
slick.bmp
Show All

Your application has been created. Please take a moment to review and adjust your application's settings.

Data Mining Tutorial

Test OAuth

Details Settings API Keys Permissions

Data Mining Tutorial
http://www.bgoncalves.com/app

Organization

Information about the organization or company associated with your application. This information is optional.

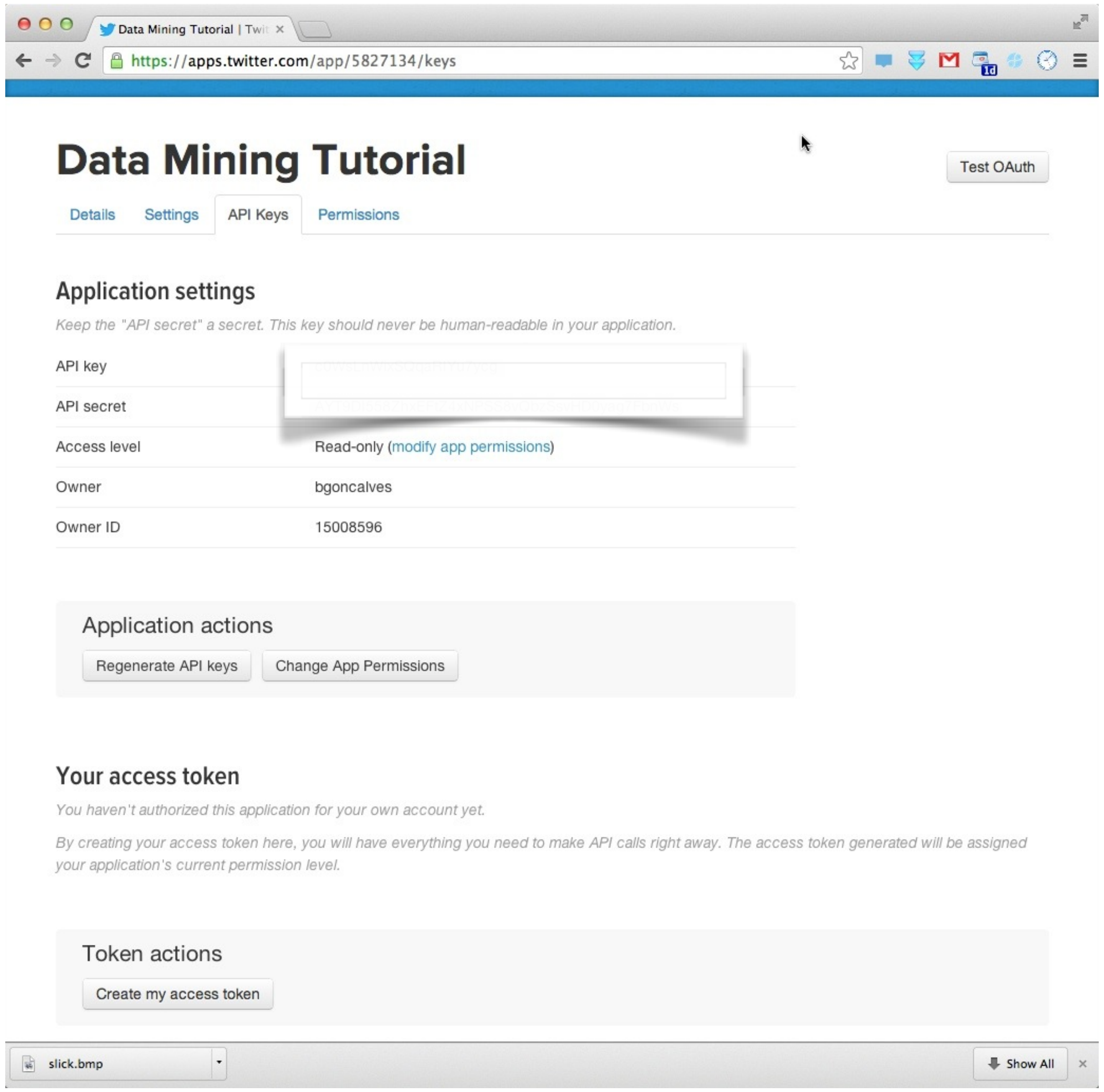
Organization	None
Organization website	None

Application settings

Your application's API keys are used to authenticate requests to the Twitter Platform.

Access level	Read-only (modify app permissions)
API key	
Callback URL	None
Sign in with Twitter	No
App-only authentication	https://api.twitter.com/oauth2/token
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token

Registering an Application



Data Mining Tutorial | Twi
https://apps.twitter.com/app/5827134/keys
Test OAuth
Details
Settings
API Keys
Permissions

Data Mining Tutorial

Application settings

Keep the "API secret" a secret. This key should never be human-readable in your application.

API key	[redacted]
API secret	[redacted]
Access level	Read-only (modify app permissions)
Owner	bgoncalves
Owner ID	15008596

Application actions

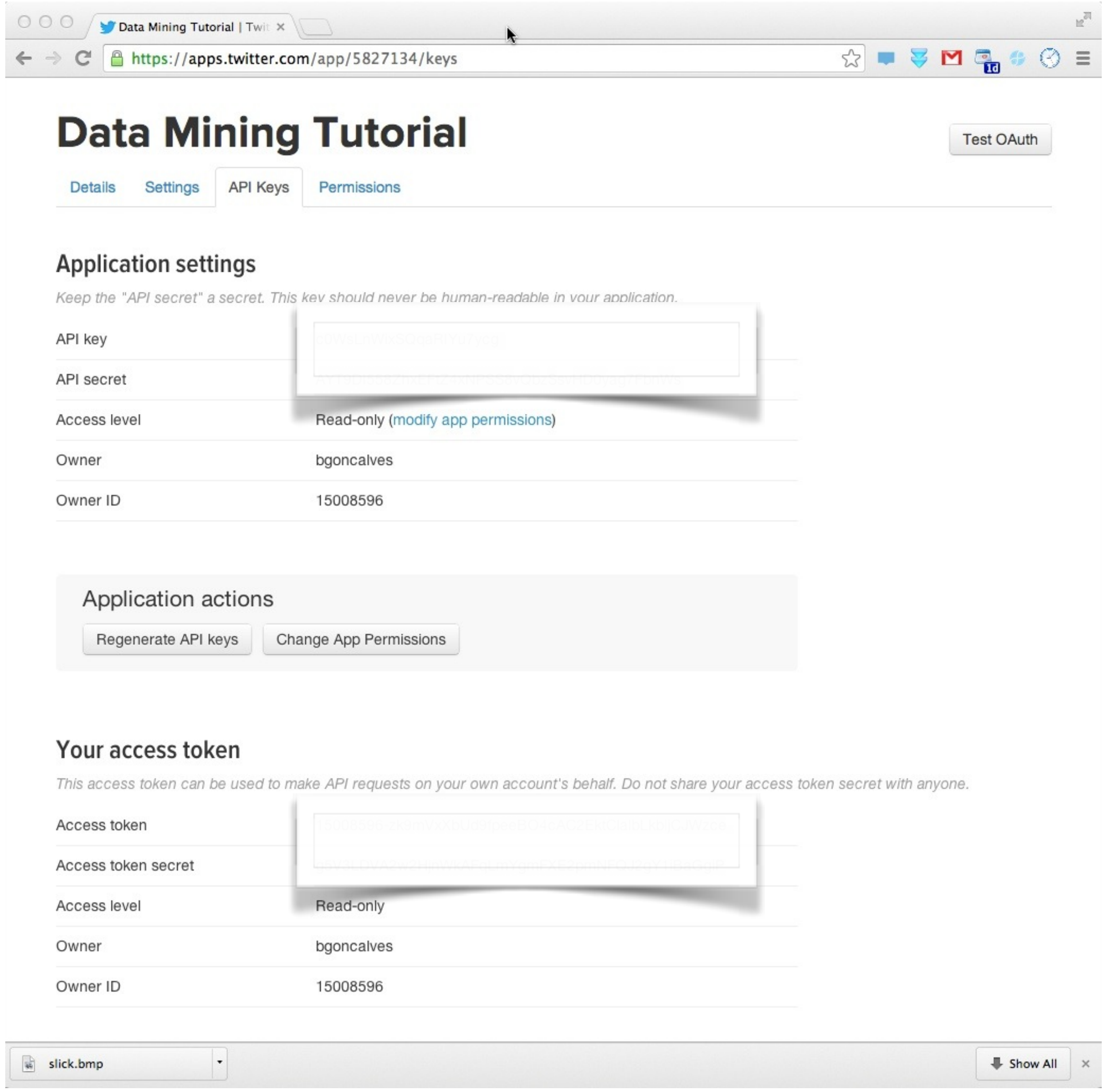
Your access token

You haven't authorized this application for your own account yet.

By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Token actions

Registering an Application


Data Mining Tutorial | Twi
https://apps.twitter.com/app/5827134/keys
Test OAuth
Details
Settings
API Keys
Permissions
slick.bmp
Show All

Data Mining Tutorial

Application settings

Keep the "API secret" a secret. This key should never be human-readable in your application.

API key	
API secret	
Access level	Read-only (modify app permissions)
Owner	bgoncalves
Owner ID	15008596

Application actions

Regenerate API keys   Change App Permissions

Your access token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

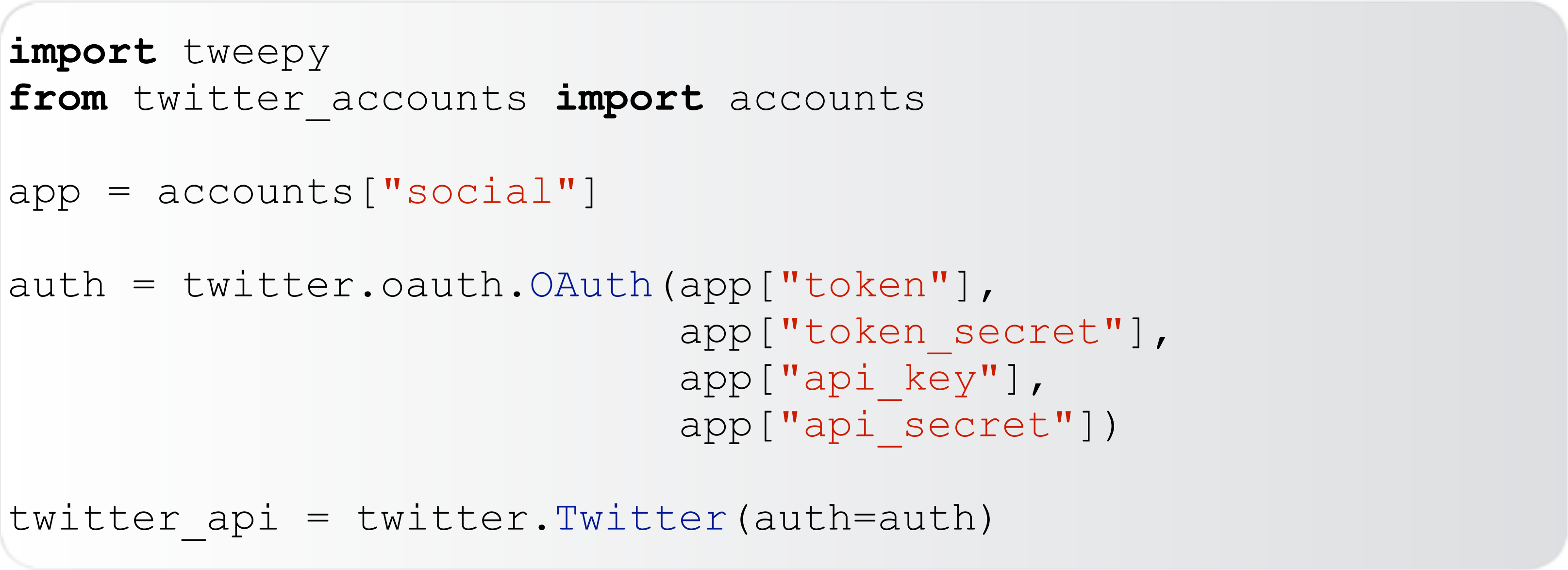
Access token	
Access token secret	
Access level	Read-only
Owner	bgoncalves
Owner ID	15008596

API Basics

https://dev.twitter.com/docs

- The twitter module provides the oauth interface. We just need to provide the right credentials.
- Best to keep the credentials in a dict and parametrize our calls with the dict key. This way we can switch between different accounts easily.
- .Twitter(auth) takes an OAuth instance as argument and returns a Twitter object that we can use to interact with the API
- Twitter methods mimic API structure
- 4 basic types of objects:
    - Tweets
    - Users
    - Entities

Authenticating with the API



```
import tweepy
from twitter_accounts import accounts

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

twitter_api = twitter.Twitter(auth=auth)
```

- In the remainder of this course, the accounts dict will live inside the twitter_accounts.py file
- 4 basic types of objects:
  - Tweets
  - Users
  - Entities
  - Places

twitter_authentication.py

Searching for Tweets

https://dev.twitter.com/docs/api/1.1/get/search/tweets

- .search.tweets(query, count)
  - query is the content to search for
  - count is the maximum number of results to return
- returns dict with a list of "statuses" and "search_metadata"

```
{u'completed_in': 0.027,
 u'count': 15,
 u'max_id': 438088492577345536,
 u'max_id_str': u'438088492577345536',
 u'next_results': u'?max_id=438088485145034752&q=soccer&include_entities=1',
 u'query': u'soccer',
 u'refresh_url': u'?since_id=438088492577345536&q=soccer&include_entities=1',
 u'since_id': 0,
 u'since_id_str': u'0'}
```

- search_results["search_metadata"]["next_results"] can be used to get the next page of results

Streaming Geocoded data

https://dev.twitter.com/streaming/overview/request-parameters#locations

- The Streaming api provides realtime data, subject to filters
- Use TwitterStream instead of Twitter object (.TwitterStream(auth=twitter_api.auth))
- .status.filter(track=q) will return tweets that match the query q in real time
- Returns generator that you can iterate over
- .status.filter(locations=bb) will return tweets that occur within the bounding box bb in real time
- bb is a comma separated pair of lon/lat coordinates.
    - -180,-90,180,90 - World
    - -74,40,-73,41 - NYC

Streaming Geocoded data

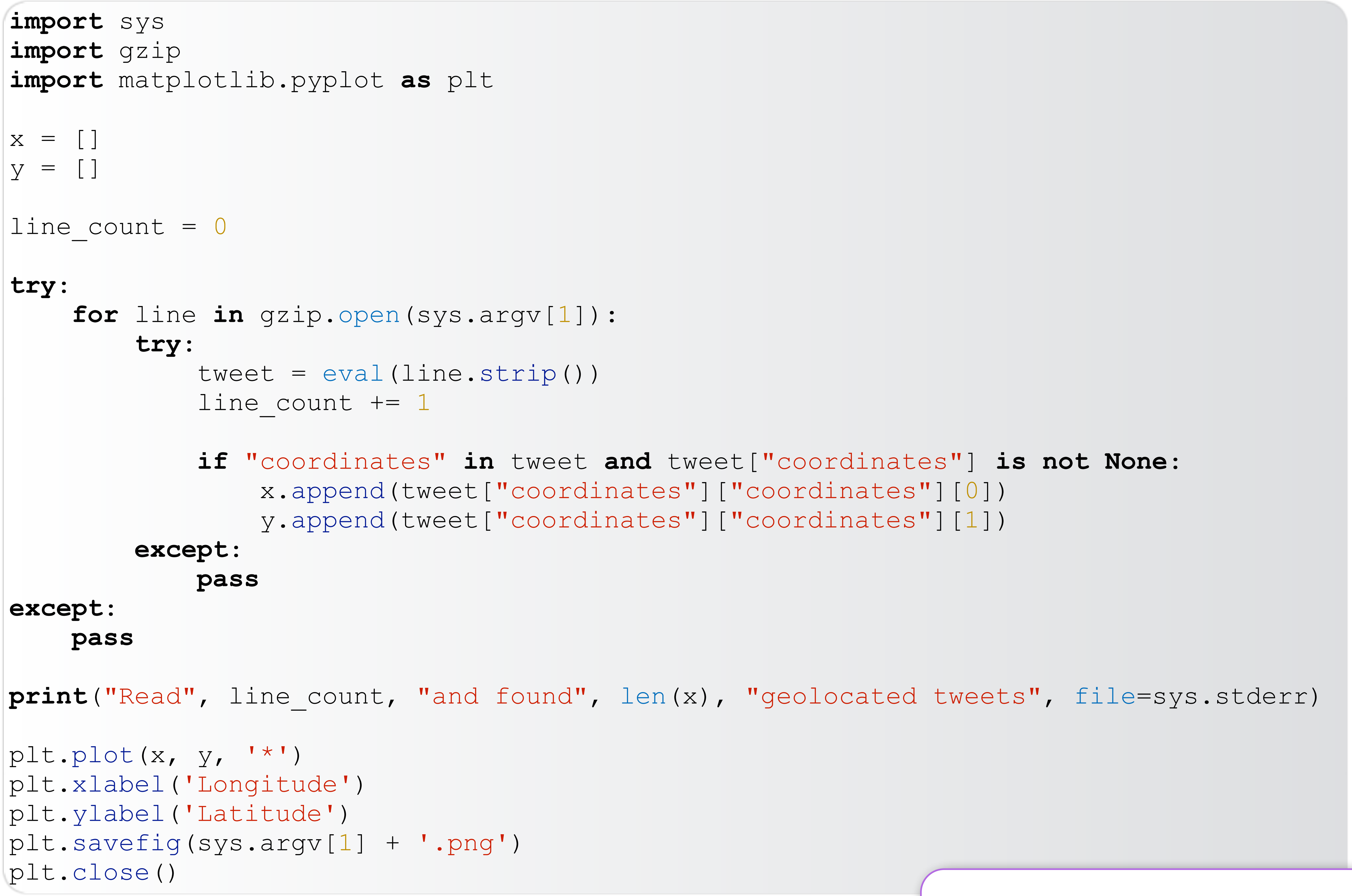
https://dev.twitter.com/streaming/overview/request-parameters#locations

```python
import twitter
from twitter_accounts import accounts
import sys
import gzip

app = accounts["social"]

auth = twitter.oauth.OAuth(app["token"],
                           app["token_secret"],
                           app["api_key"],
                           app["api_secret"])

stream_api = twitter.TwitterStream(auth=auth)

query = "-74,40,-73,41"  # NYC
stream_results = stream_api.statuses.filter(locations=query)
tweet_count = 0

fp = gzip.open("NYC.json.gz", "a")

for tweet in stream_results:
    try:
        tweet_count += 1
        print (tweet_count, tweet["id"])
        print(tweet, file=fp)
    except:
        pass

    if tweet_count % 10000 == 0:
        print(tweet_count, file=sys.stderr)
        break
```

@bgoncalves

twitter_location.py

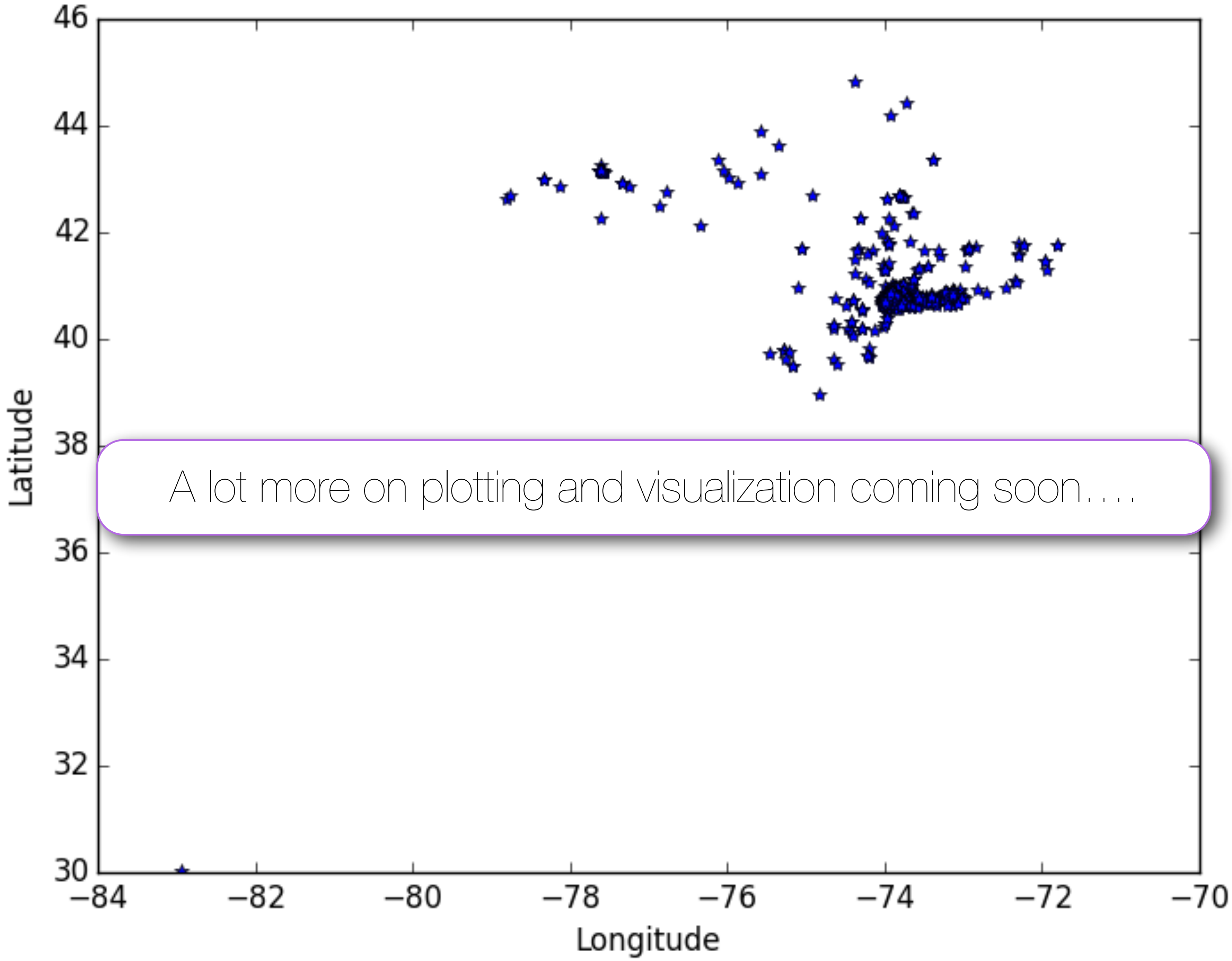
Plotting geolocated tweets



```python
import sys
import gzip
import matplotlib.pyplot as plt

x = []
y = []

line_count = 0

try:
    for line in gzip.open(sys.argv[1]):
        try:
            tweet = eval(line.strip())
            line_count += 1

            if "coordinates" in tweet and tweet["coordinates"] is not None:
                x.append(tweet["coordinates"]["coordinates"][0])
                y.append(tweet["coordinates"]["coordinates"][1])
        except:
            pass
except:
    pass

print("Read", line_count, "and found", len(x), "geolocated tweets", file=sys.stderr)

plt.plot(x, y, '*')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.savefig(sys.argv[1] + '.png')
plt.close()
```

plot_tweets.py

Plotting geolocated tweets



A lot more on plotting and visualization coming soon….

Shapefiles

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

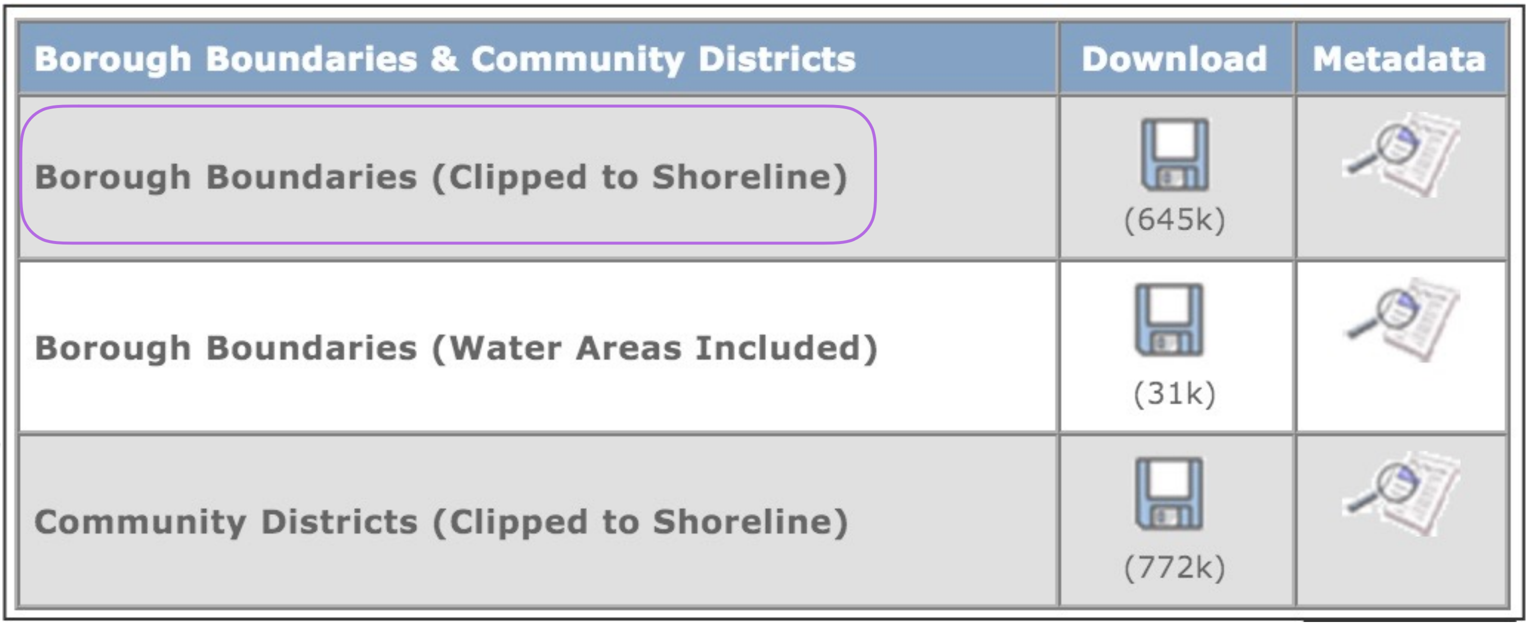
- Open specification developed by ESRI, still the current leader in commercial GIS software
- shapefiles aren't actual (individual) files…
- but actually a set of files sharing the same name but with different extensions:

```
(py35) (master) bgoncalves@underdark:$ls -l
total 4856
-rw-r--r--@ 1 bgoncalves  staff      537 Apr 17 12:40 nybb.dbf
-rw-r--r--@ 1 bgoncalves  staff      562 Apr 17 12:40 nybb.prj
-rw-r--r--@ 1 bgoncalves  staff  1217376 Apr 17 12:40 nybb.shp
-rw-r--r--@ 1 bgoncalves  staff    12905 Apr 17 12:40 nybb.shp.xml
-rw-r--r--@ 1 bgoncalves  staff      140 Apr 17 12:40 nybb.shx
-rw-r--r--  1 bgoncalves  staff      536 Apr 17 12:40 nybb_wgs84.dbf
-rw-r--r--  1 bgoncalves  staff      143 Apr 17 12:40 nybb_wgs84.prj
-rw-r--r--  1 bgoncalves  staff      257 Apr 17 12:40 nybb_wgs84.qpj
-rw-r--r--  1 bgoncalves  staff  1217376 Apr 17 12:40 nybb_wgs84.shp
-rw-r--r--  1 bgoncalves  staff      140 Apr 17 12:40 nybb_wgs84.shx
(py35) (master) bgoncalves@underdark:$
```

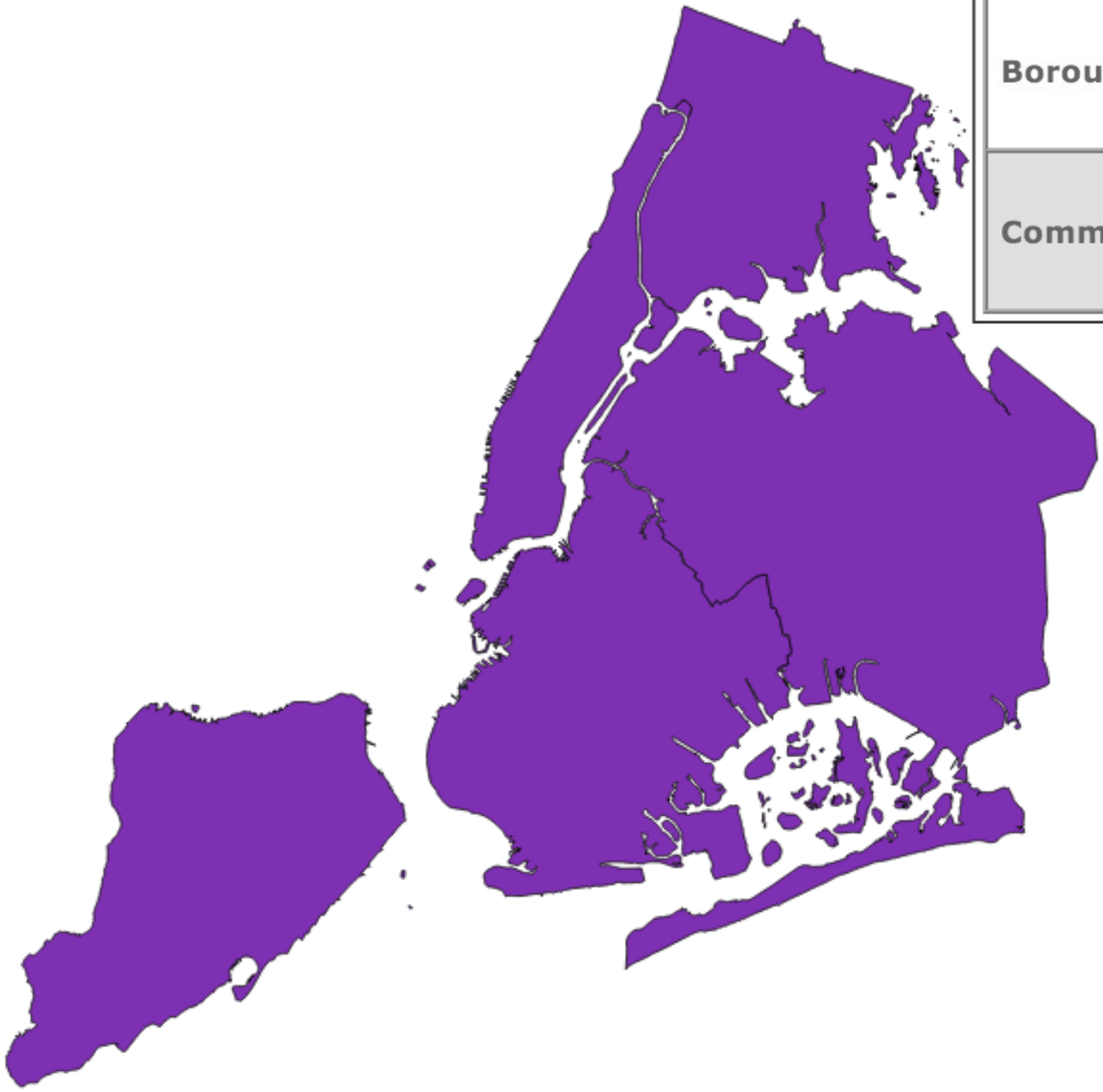
- the actual set of files changes depending on the contents, but three files are usually present:
  - .shp - also commonly referred to as "the" shapefile. Contains the geometric information
  - .dbf - a simple database containing the feature attribute table.
  - .shx - a spatial index, not strictly required

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)	🔍



- Unfortunately it doesn't use the right reference system (WGS84), so we must convert it.

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

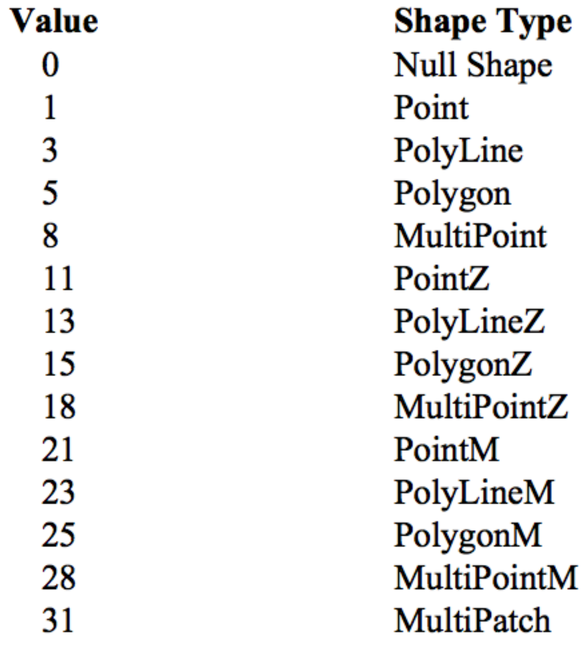
```python
import sys
import shapefile

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.

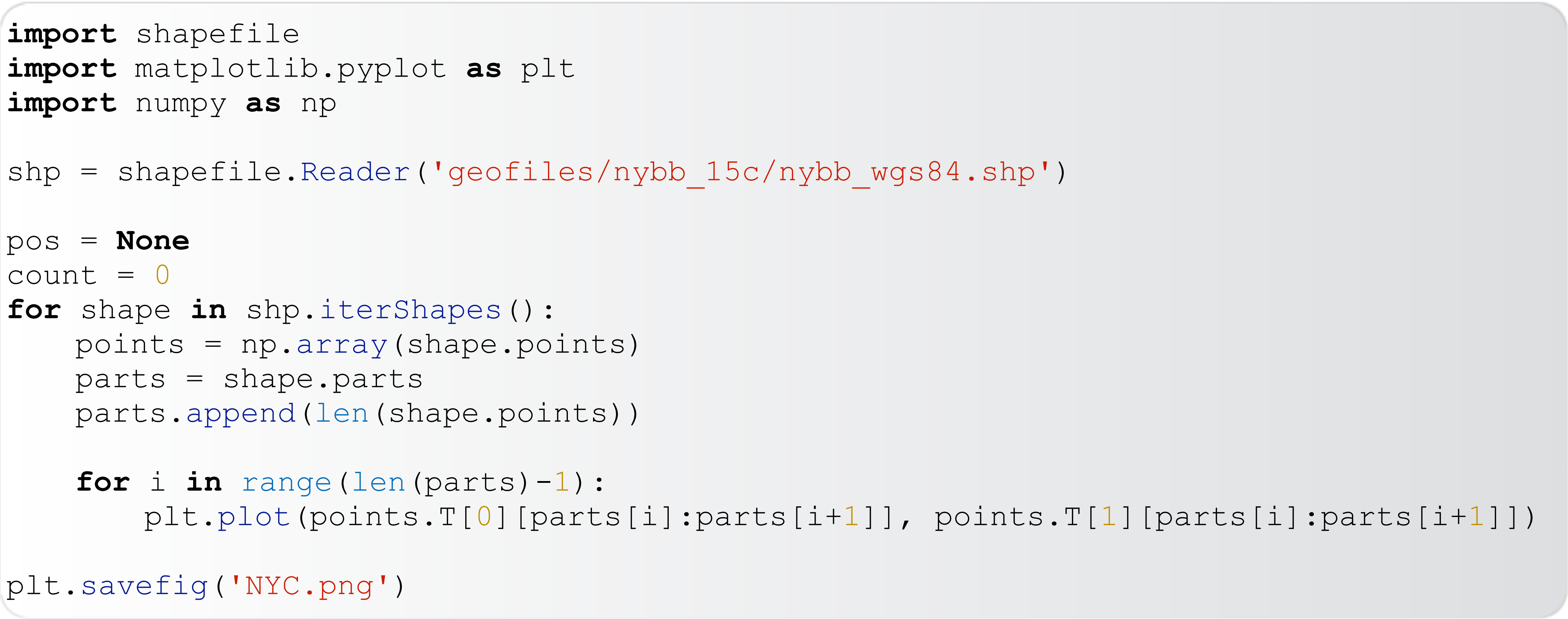


Value	Shape Type
0	Null Shape
1	Point
3	PolyLine
5	Polygon
8	MultiPoint
11	PointZ
13	PolyLineZ
15	PolygonZ
18	MultiPointZ
21	PointM
23	PolyLineM
25	PolygonM
28	MultiPointM
31	MultiPatch

pyshp

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

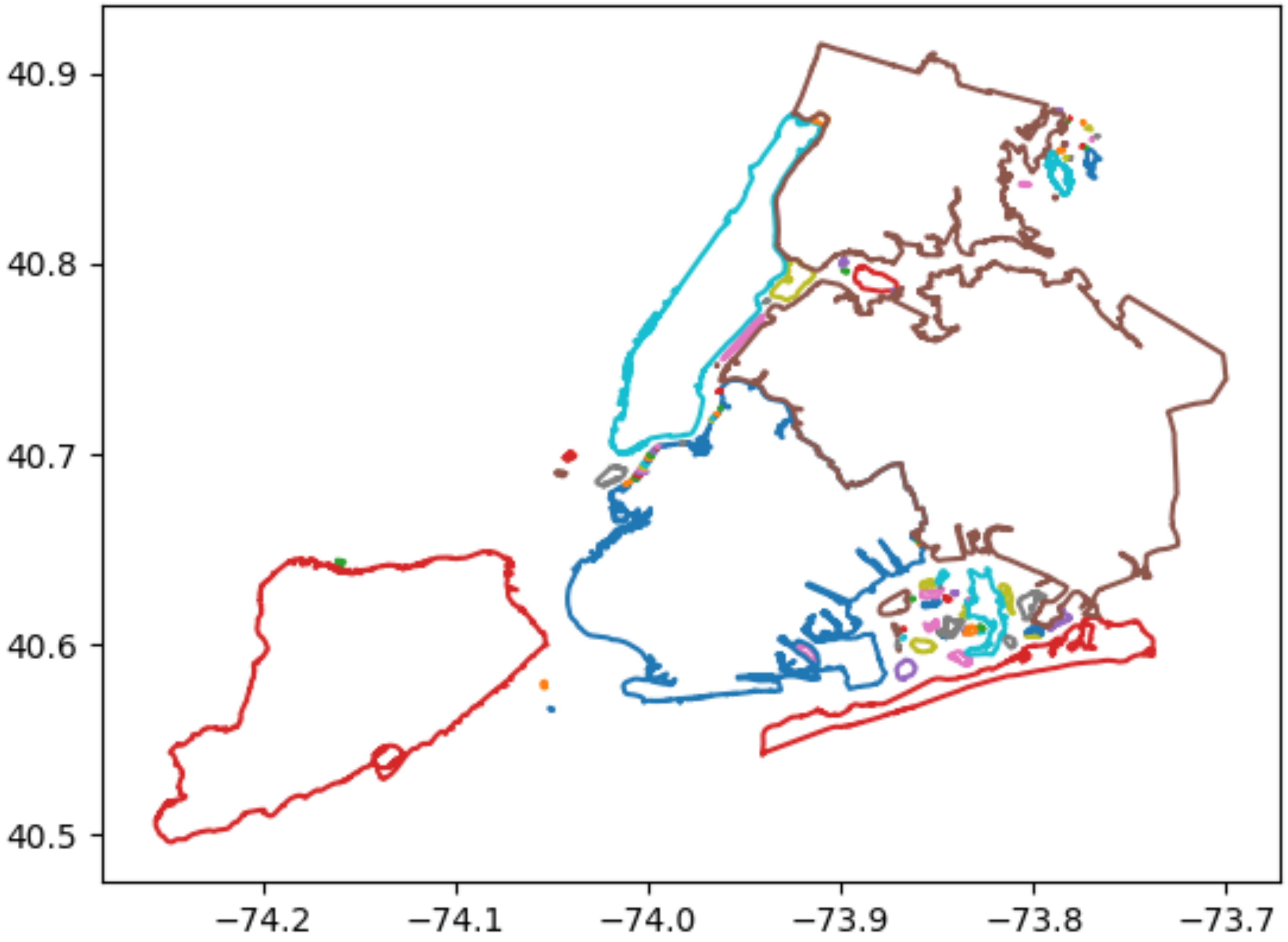
geofiles/nybb_15c/nybb_wgs84.shp



```python
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

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

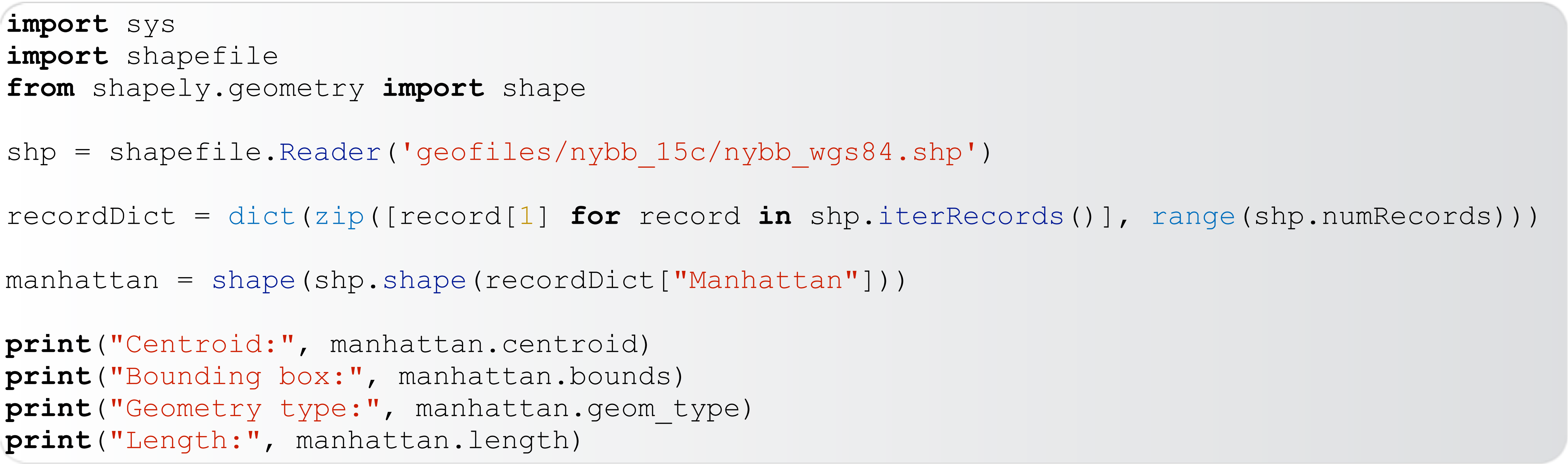
shapely

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

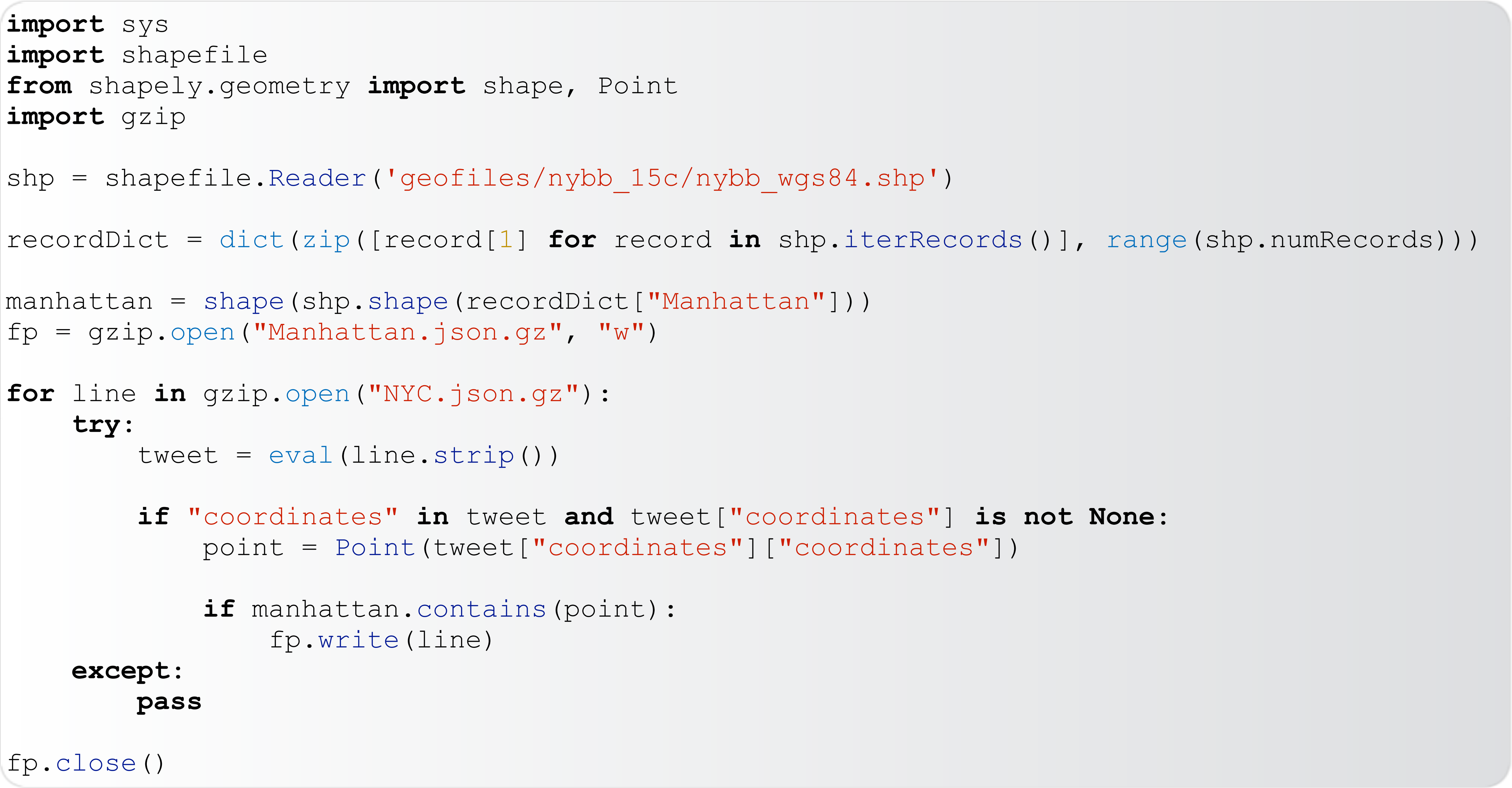
http://toblerity.org/shapely/manual.html



```python
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

Filter points within a Shapefile

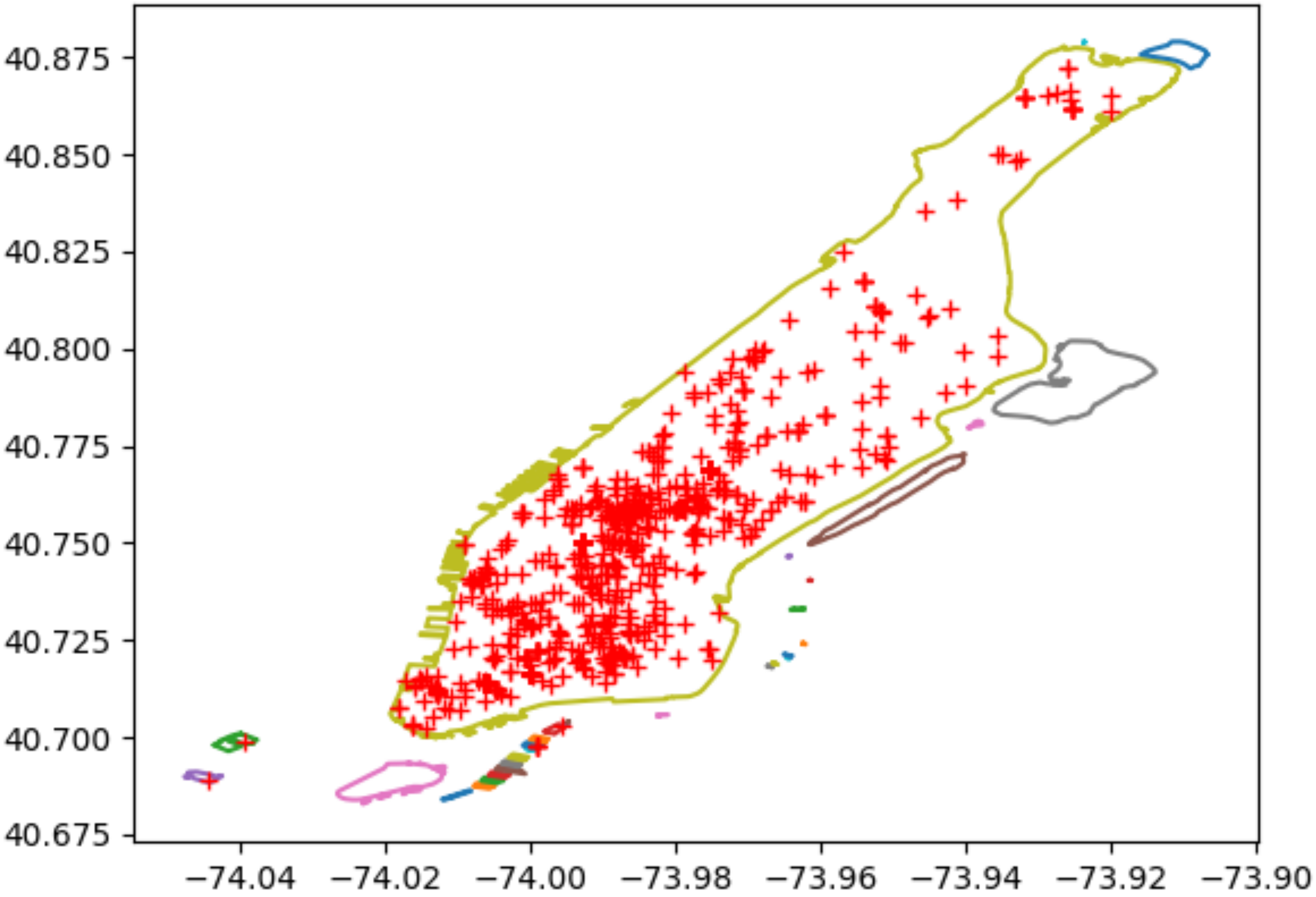


```python
import sys
import shapefile
from shapely.geometry import shape, Point
import gzip

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()
```

@bgoncalves

shapefile_filter.py

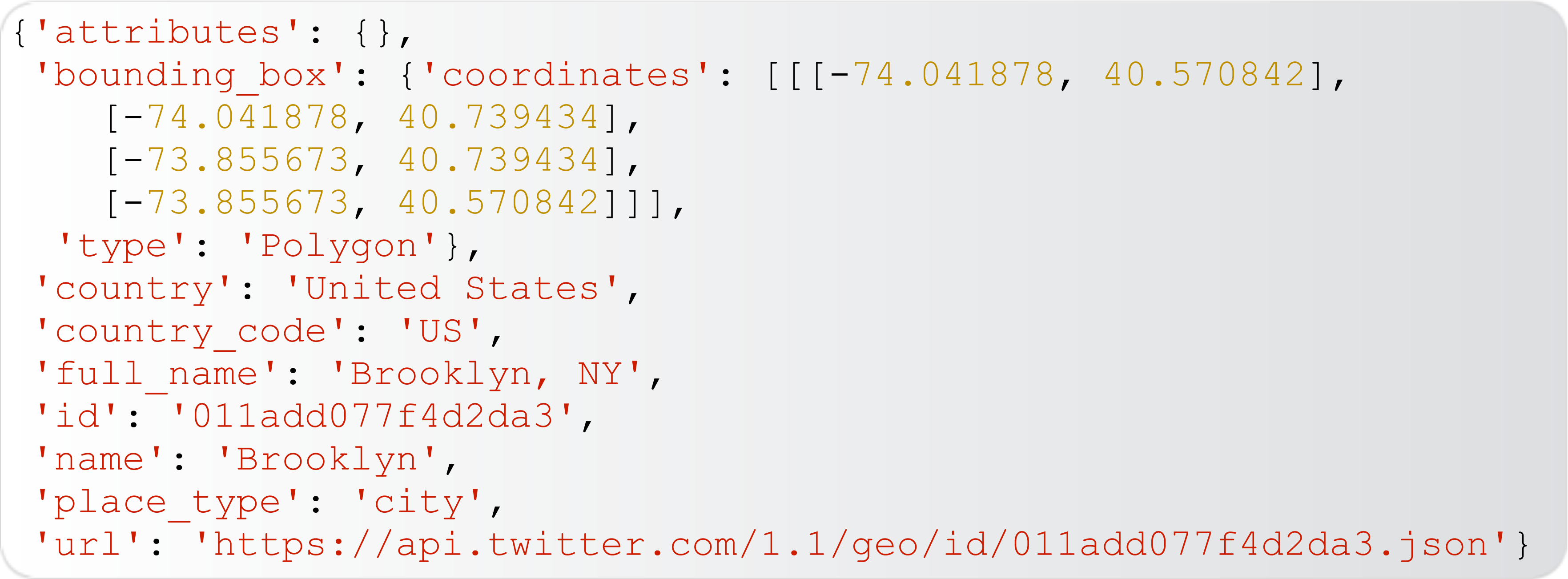
Filter points within a Shapefile



@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 already saw, 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**:

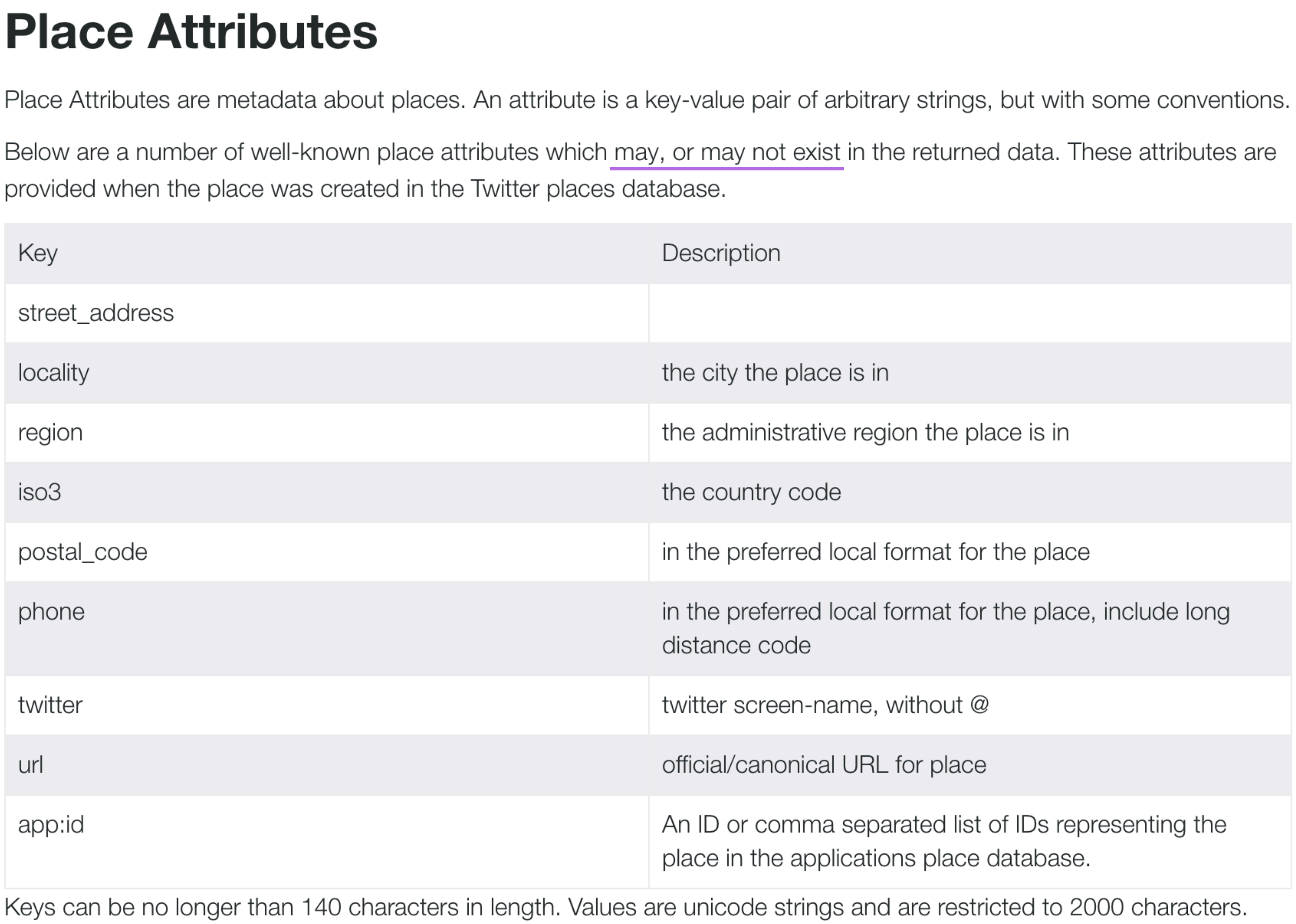
```
{'attributes': {},
 'bounding_box': {'coordinates': [[[-74.041878, 40.570842],
    [-74.041878, 40.739434],
    [-73.855673, 40.739434],
    [-73.855673, 40.570842]]],
  'type': 'Polygon'},
 'country': 'United States',
 'country_code': 'US',
 'full_name': 'Brooklyn, NY',
 'id': '011add077f4d2da3',
 '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.

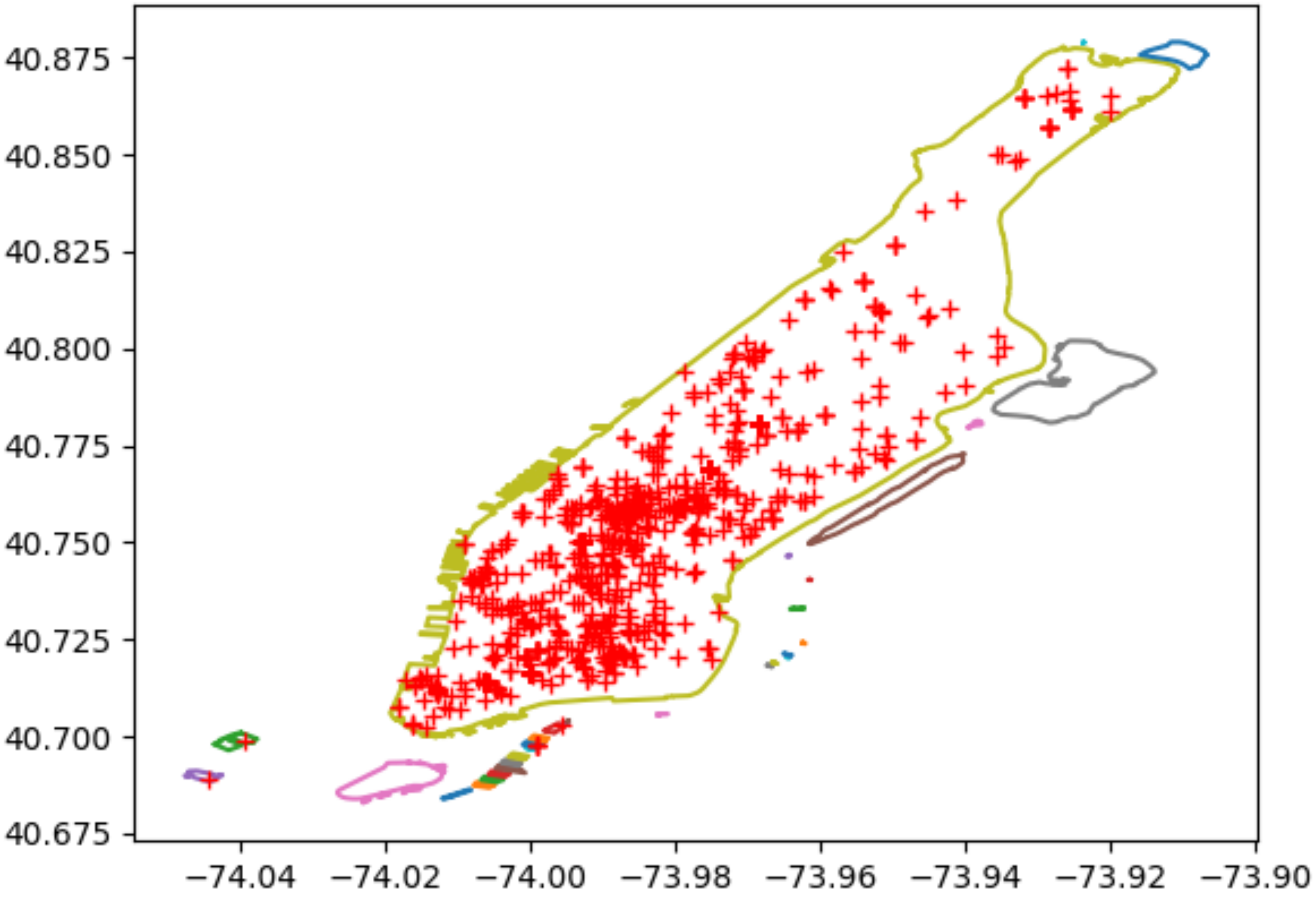
Keys can be no longer than 140 characters in length. Values are unicode strings and are restricted to 2000 characters.

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()
```

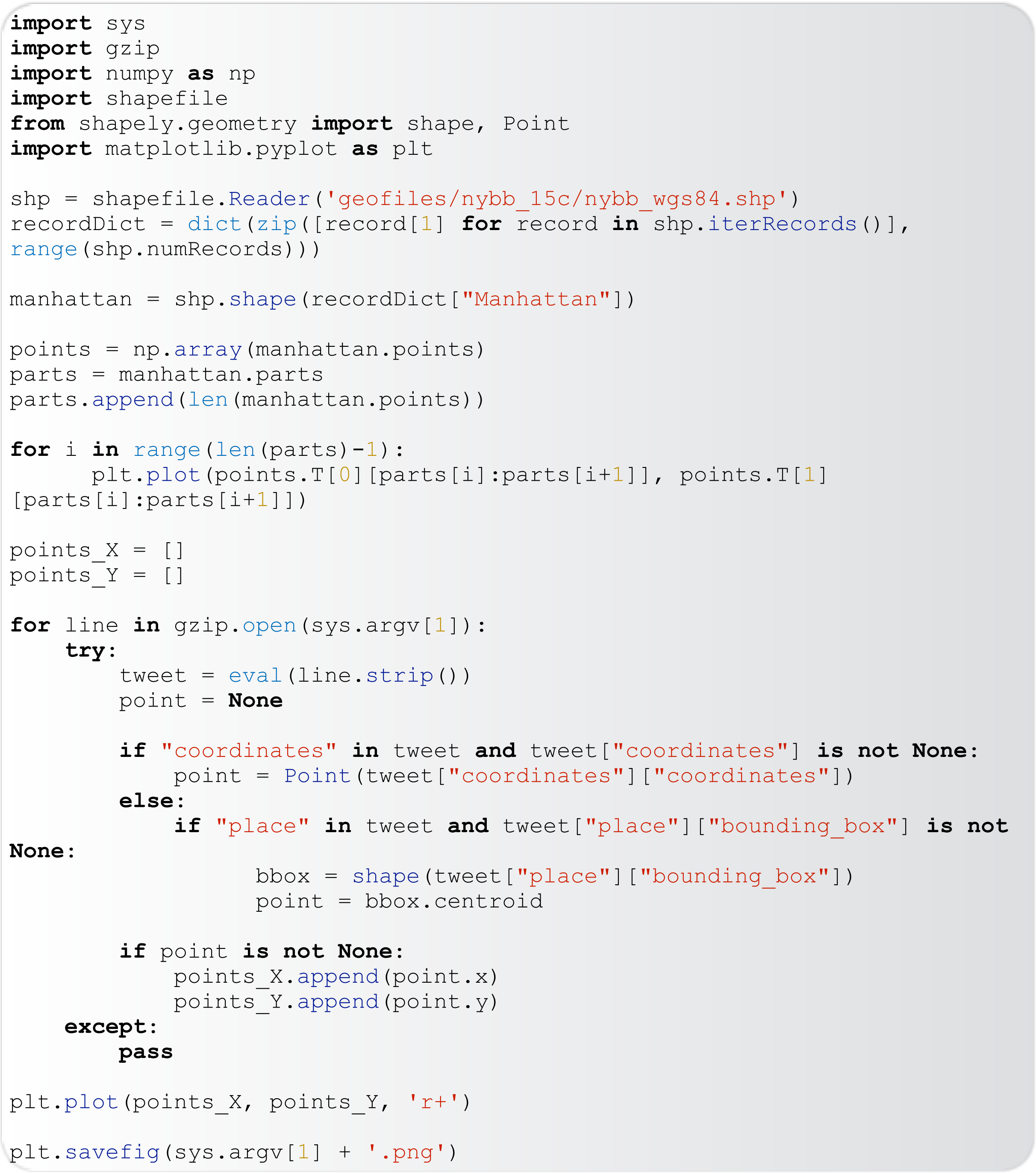
shapefile_filter_places.py

Filter points and places



@bgoncalves

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

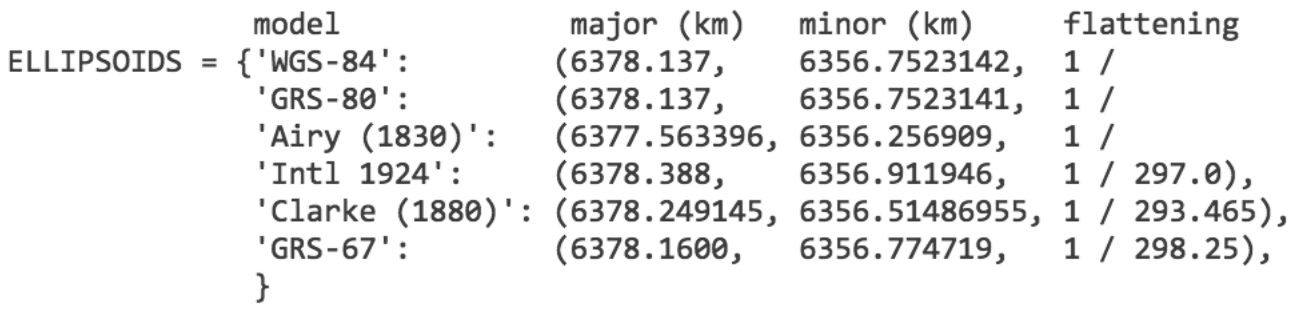
https://geopy.readthedocs.io/en/1.10.0/

- 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),
              }
```

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

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

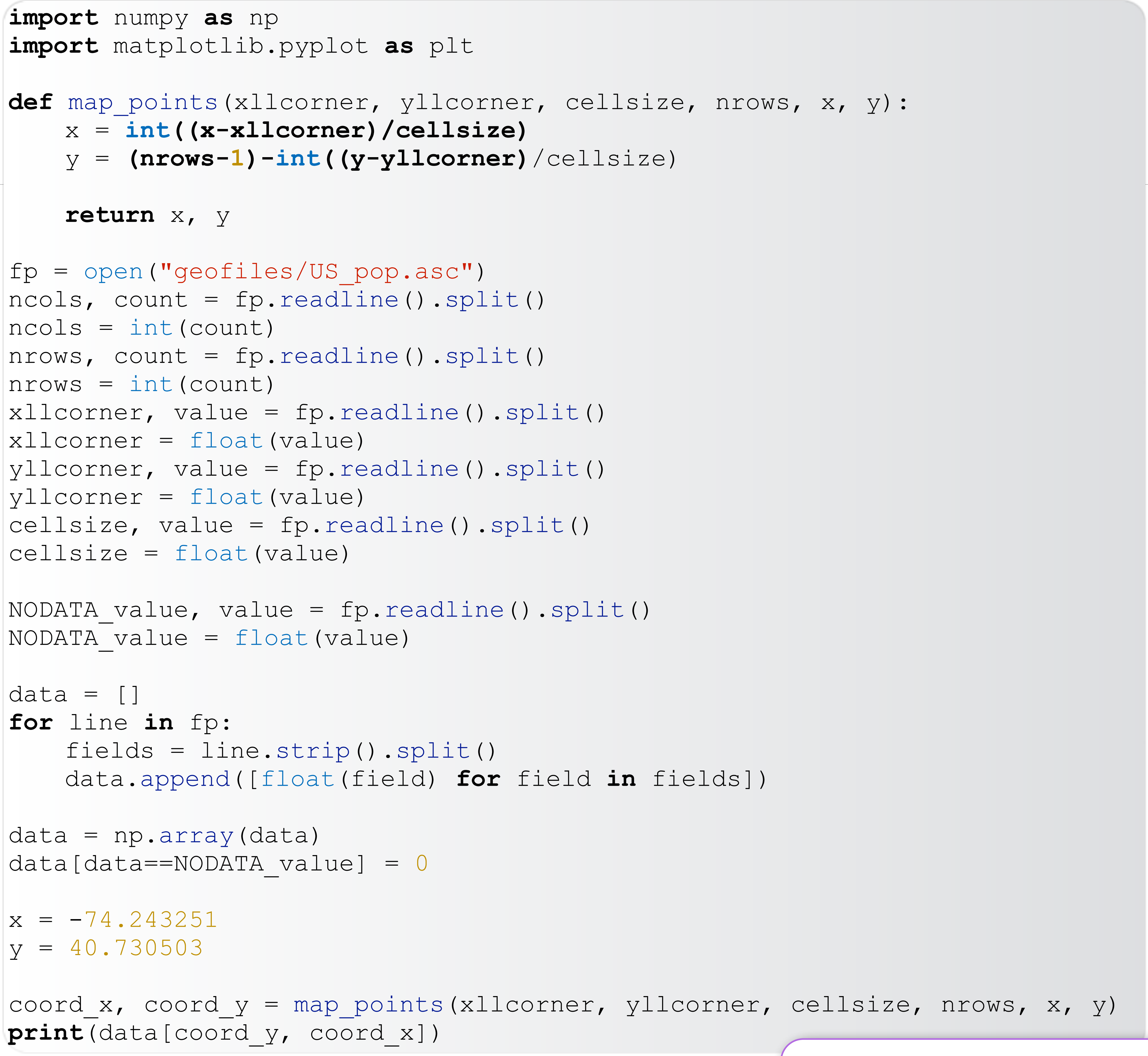
geofiles/US_pop.asc

ASCII Grid

```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

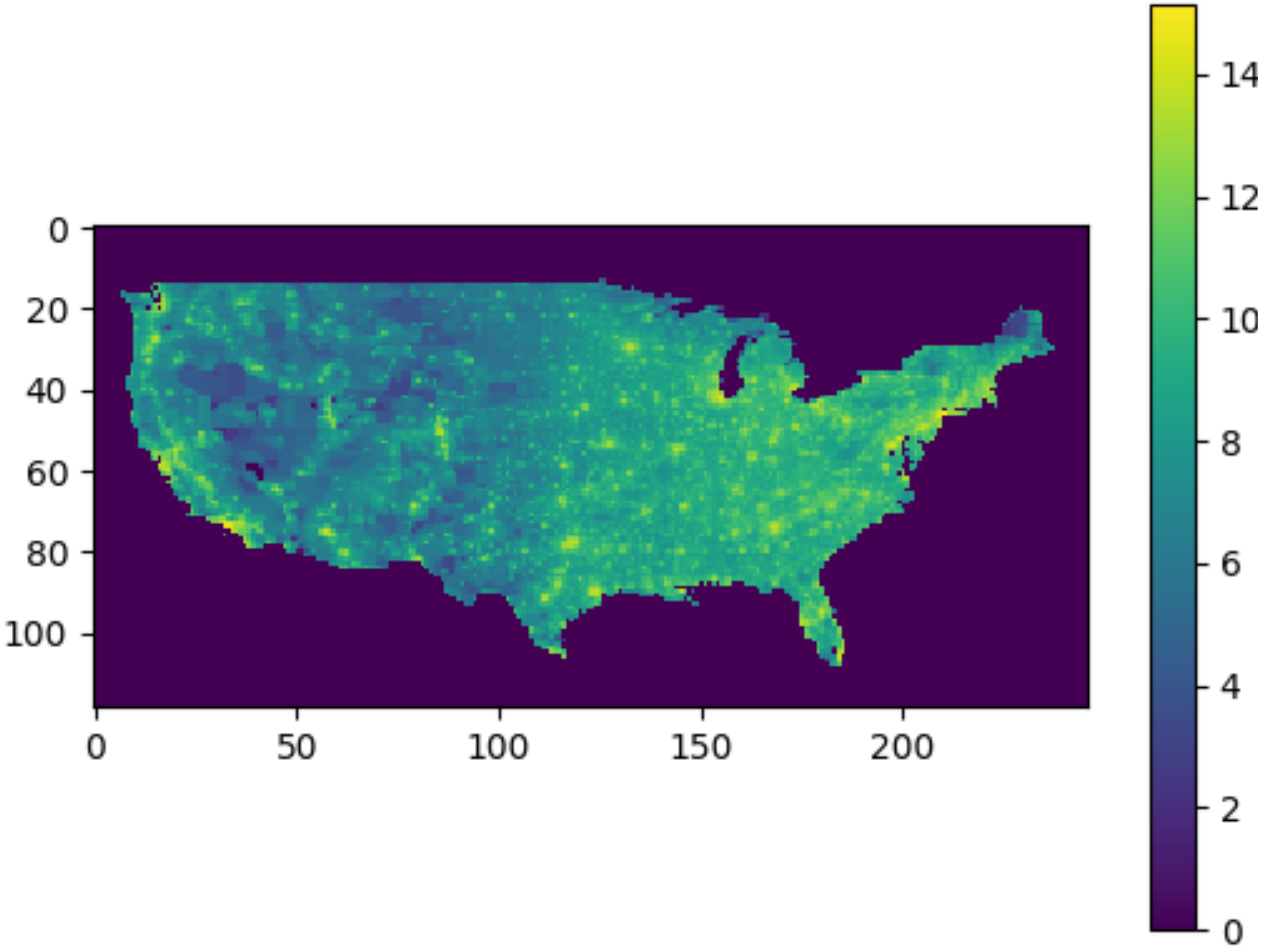
ASCII Grid



```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

ASCII Grid

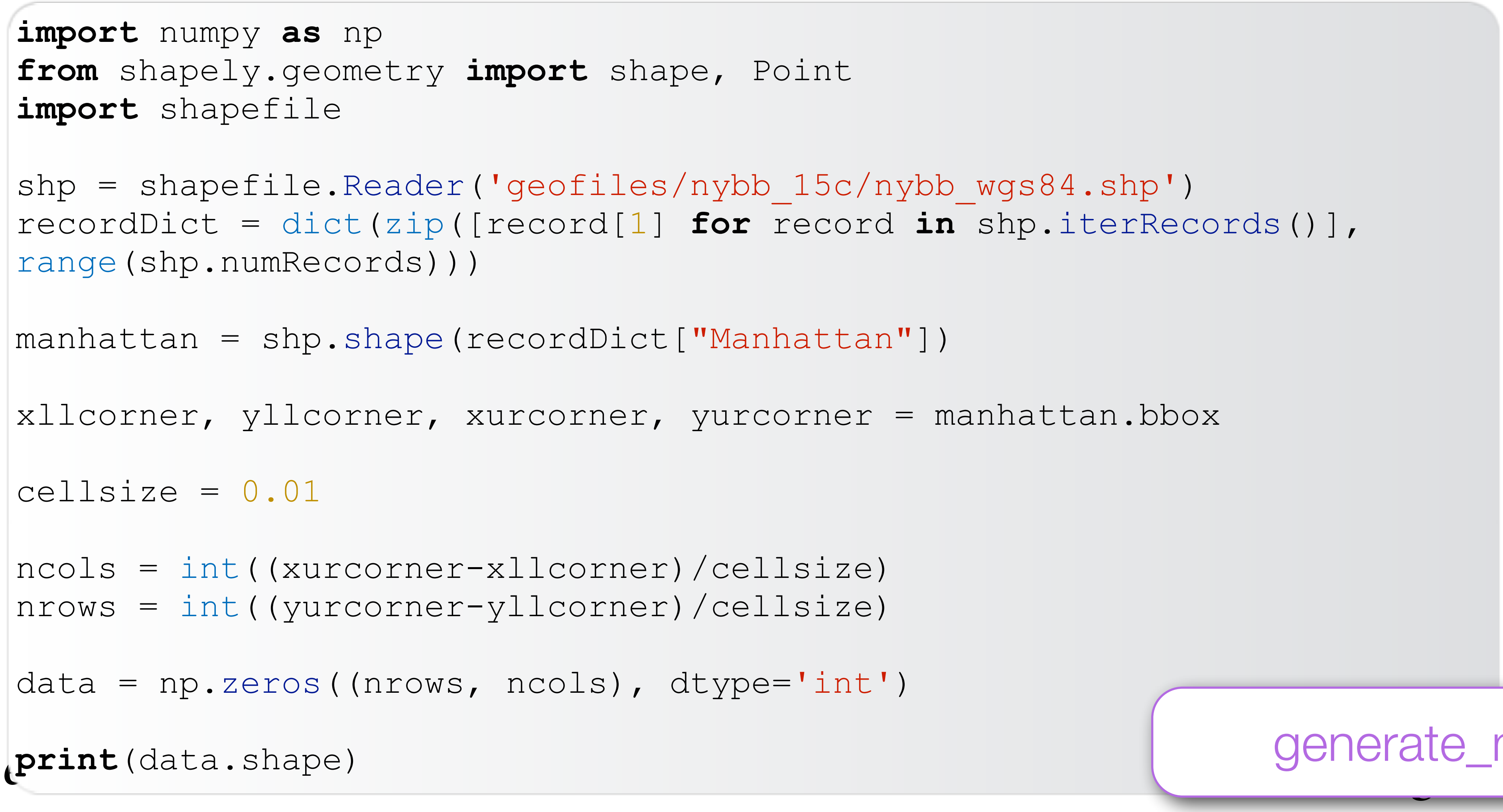


@bgoncalves

asc_load.py

ASCII Grid

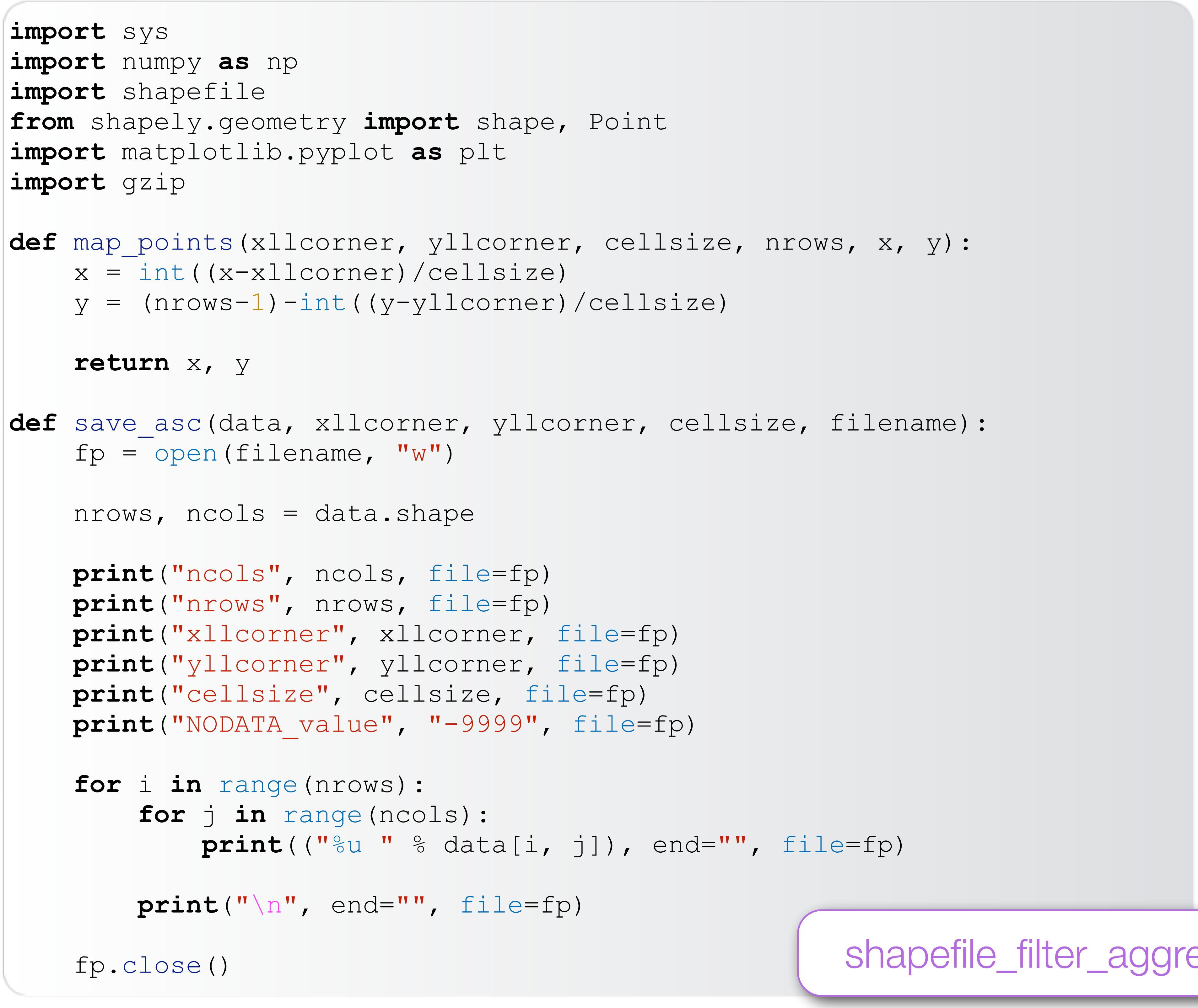
- This type of grid is a very convenient way to aggregate spatial data.
- Simply map **lat, lon** pairs to matrix entries and then increment the values
- All we need is to define the **bbox** we are interested in, and the size of each cell and create a matrix with that shape.



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

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"])

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)
```

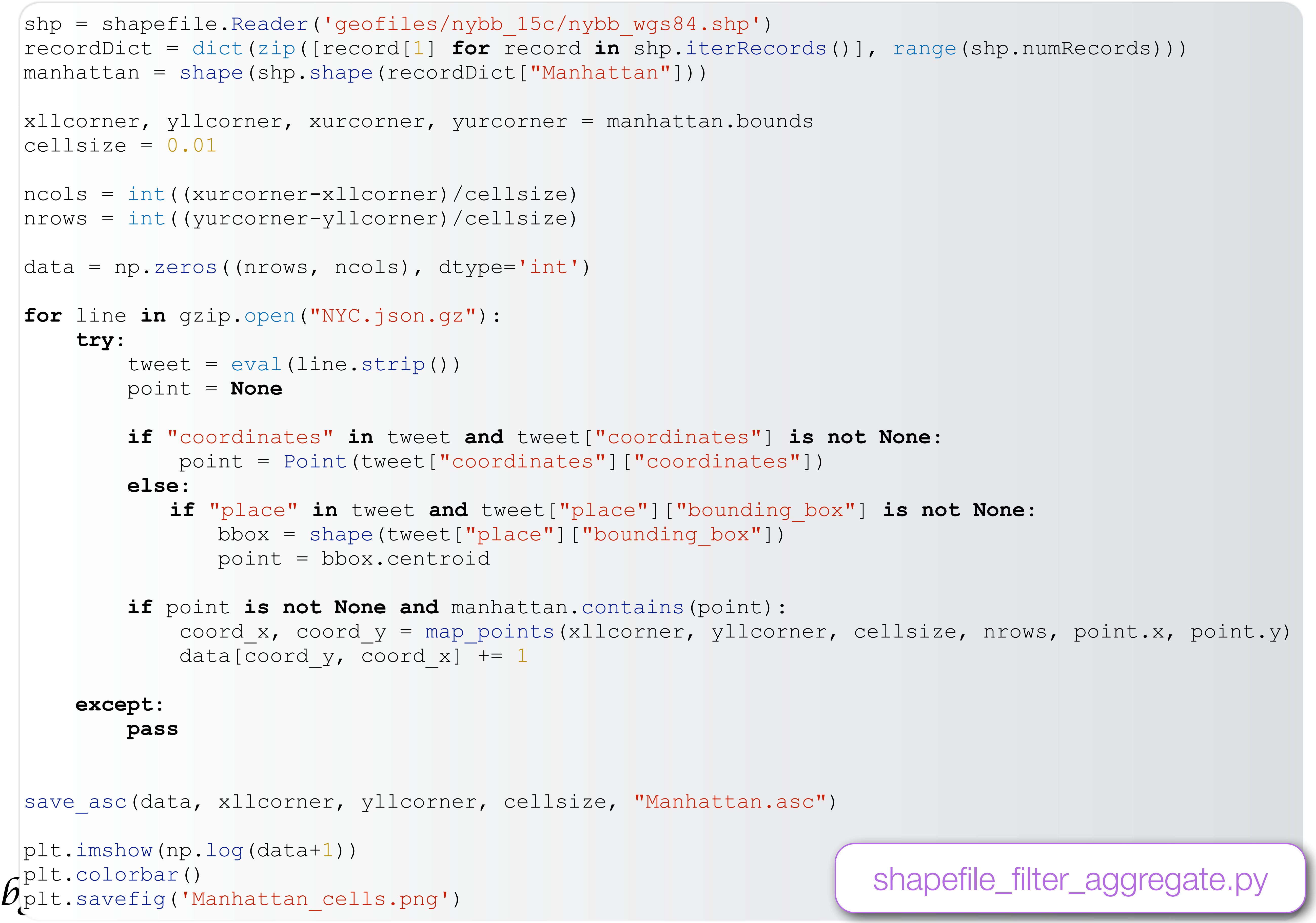
generate_matrix.py

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

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()
```

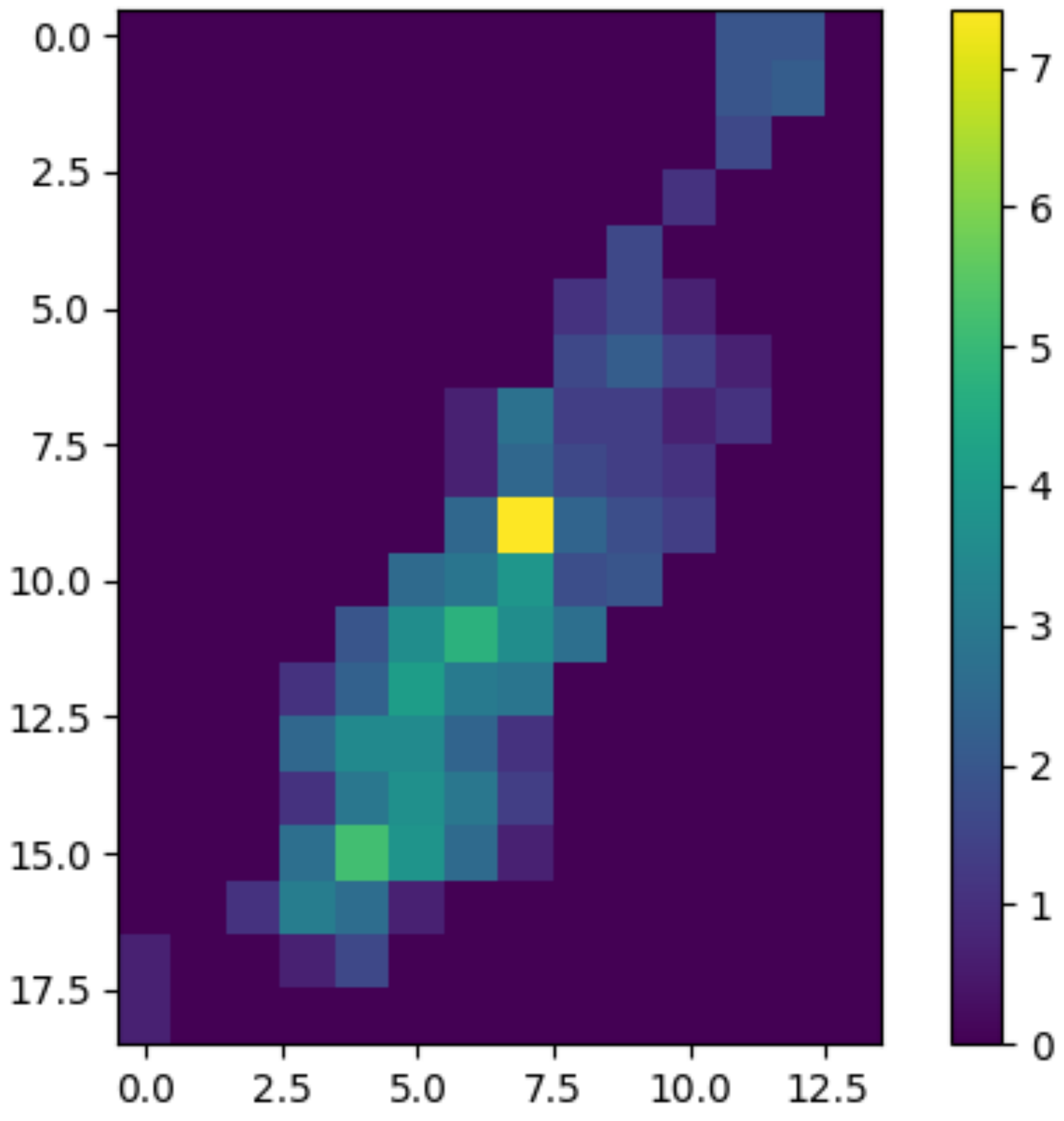
shapefile_filter_aggregate.py



```
shp = shapefile.Reader('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("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

Aggregate


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
2.5
5.0
7.5
10.0
12.5
7
6
5
4
3
2
1
0