CE236N_Problem2_Hill

October 6, 2016

0.0.1 L. Drew Hill (SID 21707129)

0.0.2 Note: I had no partners... I didn't want to subject them to my non-engineering background.

0.1 Task 1

Fiona is an OGR-based Python library that is used to read and write geometry data types like multi-layered GIS formats and zipped virtual file systems via standard Python input-output style. It can be used to convert a geometric database into a postgreSQL-compatible database.

Shapely is used for the manipulation and spacial analysis in Python of planar geometric objects (which can be read and written via Fiona) that are not tied to specific data formats or coordniate systems. Shapely is based on GEOS and is very fast and efficient with planar analyses like geometric buffers, unions, intersections, and centroids.

PostGIS can perform both spacial and geometric analysis and read/write geometry data types to a PostgreSQL database management system. PostGIS is based on light-weight geometries, reducing disk and memory footprints and providing considerable convenience for speedily querying large spatial databases in long-term PostgreSQL storage.

Simple ad-hoc spatial data analysis could probably be done efficiently and quickly using a combination of Fiona to read/write the data and Shapely to analyze it. Longer term storage involving tidy data structure and relational storage would be best performed using PostGIS and postgreSQL, perhaps through the psycopg2 Python package if necessary. For data distribution, I would employ Fiona to format and read data out of Python in a way that would produce minimal formatting issues for folks wanting to import the data into Python, ESRI or another analysis tool.

0.2 Task II

0.2.1 Part 1

```
In [1]: import psycopg2
        import json
        import random
        from dateutil.parser import parse
        from dateutil.parser import parse
```

create database via psql in terminal
CREATE DATABASE task2 OWNER Lawson TEMPLATE template_postgis ENCODING utf8;
** start server in shell**
pg_ctl -D /usr/local/var/postgres -l /usr/local/var/postgres/server.log start

```
In [24]: # connect to database that I created
         try:
             conn =
             psycopg2.connect("dbname=task2 user=Lawson host=localhost password=***
         except:
             print "Cannot connect to database"
         cur = conn.cursor()

         # # verify error-free execution
         # cur.execute("""SELECT srtext FROM spatial_ref_sys WHERE srid = 32610;""
         # rows = cur.fetchall()
         # for row in rows:
         #     print "", row[0]

         # conn.commit()
         # conn.close()
```

0.2.2 Part 2

```
In [11]: ## Import the twitter database
         with open('tweets_1M.json','r') as f:
             twitters=json.load(f)

         ## Randomly select 100,000 tweets
         k = 100000
         random_index = random.sample(range(len(twitters)), k)
         # reorder the index, then take each indexed object from the list
         tweets = [twitters[i] for i in sorted(random_index)]

In [12]: ## create ST_GeomFromText object for use in SQL
         for i in range(0,len(tweets)):
             tweets[i]['geoloc'] = 'Point(%s' % tweets[i]['lng'] +
             ' %s)' % tweets[i]['lat']

In [13]: # parse datetime
         for i in range(0,len(tweets)):
             tweets[i]['timeStamp'] = parse(tweets[i]['timeStamp'])

In [ ]: # # create empty geoJSON object
        # tweet_geo = {
        #     "type": "FeatureCollection",
        #     "features" : []
        # }

        # # fill geoJSON object with each tweet
        # for i in range(0,len(tweets)):
        #     twit = [
        #         {"type": "Feature",
        #             "geometry": {
```

```
#                 "type": "Point",
#                 "loc": [tweets[i]['lat'],tweets[i]['lng']]
#                  },
#                  "properties": {
#                      "id": tweets[i]['id'],
#                      "text": tweets[i]['text'],
#                      "time": tweets[i]['timeStamp'],
#                      "user_id": tweets[i]['user_id']
#                  }
#              }]
#     tweet_geo['features'].append(twit)

# # # Save geo data as geoJSON
# # with open('tweet_geo.json', 'w') as fout:
# #     fout.write(json.dumps(tweet_geo, indent=4))
```

0.2.3 Part 3

```
In [ ]: ## determine max length of text and thus what CHAR(n) to use
        text_len = []
        for i in range(0,len(tweets)):
            a = [len(tweets[i]['text'])]
            text_len.append(a)

        max(text_len)
```

```
In [14]: ## Create table in which to store tweets
         cur.execute("""CREATE TABLE tweets (id char(50) PRIMARY KEY,
             userid INT, loc CHAR(100), time timestamptz, text CHAR(506));""")
```

```
In [15]: ## Insert tweets-- each as a new row
         cur.executemany("""INSERT INTO tweets(id, userid, loc, time, text)
             VALUES (%(id)s, %(user_id)s, ST_GeomFromText(%(geoloc)s, 4326),
             timestamptz(%(timeStamp)s), %(text)s)""", tweets)
         # commit new datatable to database
         conn.commit()
```

0.2.4 Parts 4 & 5

The census population shapefile data for counties in California were downloaded from http://www.census.gov/cgi-bin/geo/shapefiles2010/main, and inserted into my database (called "task2").

Convert shape files into form compatible with my database in SHELL:

shp2pgsql -I -W "latin1" -s 4326 /Users/Lawson/Box Sync/Current Coursework/CE263 - Scalable Spatial Analytics/Assignments/Problem 2/tl_2010_06_county10/tl_2010_06_county10.shp public.ca_census_tract | psql -d task2

0.2.5 Part 6

From the CA census shapefile, I was able to establish how many of the 100k tweets were tweeted from Contra Costa county. The "ST_Interects" command was used to isolate all tweet coordinate pairs inside of the shapefiles presented to PSQL, and a select command was used to ensure that only the Alameda County shapefile was presented. A count command was used to count all of these Contra-Costa-specific tweet IDs.

*A total of **8,621 tweets** from my 100k subset originated from Contra Costa County*

```
In [9]: ## Calculate the # of tweets in Contra Costa county
        cur.execute("""SELECT COUNT(tweets.id)
        FROM tweets, ca_census_tract
        WHERE ST_Intersects(ca_census_tract.geom, tweets.loc)
        AND ca_census_tract.name10 = 'Contra Costa';""")

        cur.fetchall()

Out[9]: [(86L,)]
```

0.2.6 Part 7

I next set about counting the number of tweets sent from within 100 miles outside of the Alameda county border. First, a 160,934 meter (100 mile) buffer was set around the geographic border of Alameda County. This buffer was converted into a geometry and then any tweet locations intersecting the buffer. Next, all tweets that also interesected with the geometry of the county shape (i.e. tweets coming from within the county) were excluded. All remaining tweets were counted by unique tweet id.

Approximately **68,320 tweets** of the 100k randomly selected tweets were sent from within outside 100 mi of Alameda County. This makes sense considering the major metropolitan centers included within a 100 mi radius of Alameda county's borders).

```
In [31]: ## Calculate how many tweets are w/in 100 mi outside of Alameda county

         ## First get all points within 100.00 m of any shapefile border
         ## Second, makesure PSQL is only considering the Alameda shapefile
         ## Third, exclude all points within the county polygon itself,
         ## thus leaving only points  100m outside the border.

         cur.execute("""SELECT COUNT(tweets.id)
         FROM tweets, ca_census_tract
         WHERE NOT ST_Intersects(ca_census_tract.geom, tweets.loc)
         AND ST_Intersects(
         ST_Buffer(ca_census_tract.geom::Geography,160934)::geometry, tweets.loc)
         AND ca_census_tract.name10 = 'Alameda';""")

         cur.fetchall()


         # # attempt using degrees and a wonky conversion factor
```

```
# cur.execute("""SELECT count(tweets.id)
# FROM tweets, ca_census_tract
# WHERE NOT ST_Intersects(ca_census_tract.geom, tweets.loc)
# AND ST_DWithin(ca_census_tract.geom, tweets.loc, 1.59)
# AND ca_census_tract.name10 = 'Alameda' ;""")

# cur.fetchall()
```

0.2.7 Part 8

In [30]:
```
## Create table in which to store 2010 census pop data
cur.execute("""CREATE TABLE ca_census
(GEOID integer PRIMARY KEY,
SUMLEV varchar(3),
STATE varchar(2),
COUNTY varchar(3),
CBSA varchar(5),
CSA varchar(3),
NECTA integer,
CNECTA integer,
NAME varchar(30),
POP100 numeric,
HU100 integer,
POP100_2000 integer,
HU100_2000 integer,
P001001 integer,
P001001_2000 integer);""")

# commit change to database
conn.commit()
```

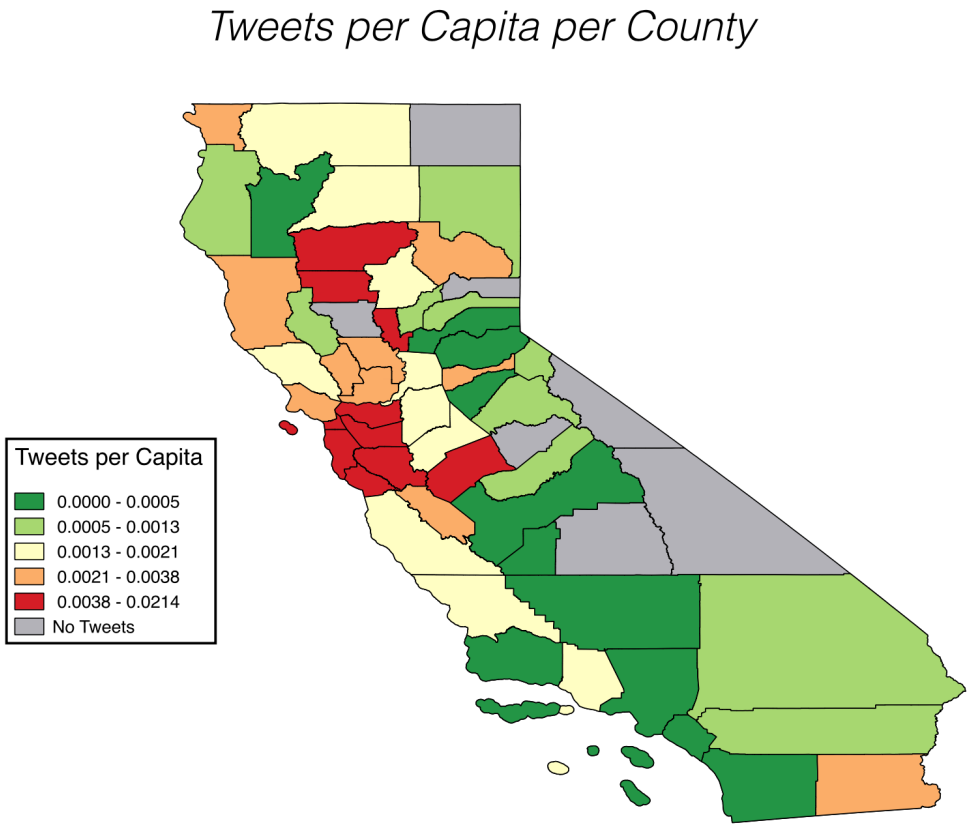
In [31]:
```
# Create SQL command to insert CSV into table "ca_census":
copy_sql_string = """ COPY ca_census FROM stdin
WITH CSV HEADER DELIMITER as ',' """

# open csv file
f = open(r'/tl_2010_06_county10/ca_census.csv', 'r')
# copy csv file
cur.copy_expert(sql=copy_sql_string, file=f)
# commit changes
conn.commit()

# close CSV
f.close()
```

0.2.8 Part 9

Tweet per capita was calculated for each county using the code outlined below. Briefly, I joined the census shapefiles with population values (as matched by geoid, or county id) into a temporary

table, joined that temporary table with the table of all 100k tweets, and then counted the total number of tweets divided by county population as grouped by county.

These data were stored in a table along with all ocunty shapefiles. This table was used to produce a QGIS image depicting tweets per capita using a red-green heatmap-style shading, as seen below.



In [ ]:
```
### Some prep work

    # convert geoid10 column of census_tract to match INTEGER type of the geoi
    cur.execute("""ALTER TABLE ca_census_tract ALTER COLUMN geoid10
    TYPE integer USING (geoid10::integer);""")
    conn.commit()

    # rename geoid10 to match ca_census geoid
    cur.execute("""ALTER TABLE ca_census_tract RENAME COLUMN
        geoid10 to geoid;""")
    conn.commit()

    # set ca_census_tract primary key to geoid
        # drop current key
    cur.execute("""ALTER TABLE ca_census_tract DROP CONSTRAINT
        ca_census_tract_pkey;""")
```

```
        # make geoid primary key
    cur.execute("""ALTER TABLE ca_census_tract DROP CONSTRAINT
        ca_census_tract_pkey;""")
    conn.commit()

    cur.execute("""CREATE TABLE tweet_percap_final
    (geoid INT PRIMARY KEY, pop INT, geom geometry ,
        tweet_percap numeric);""")
    conn.commit()
```

In [9]:
```
# Visualize tweets per capita for CA counties using a single query
cur.execute("""WITH newtable AS (
    SELECT ca_census.geoid geoid, ca_census.pop100 pop,
        ca_census_tract.geom geom
    FROM ca_census, ca_census_tract
    WHERE ca_census.geoid = ca_census_tract.geoid
    )
    INSERT INTO tweet_percap_final
    SELECT newtable.geoid, newtable.pop, newtable.geom,
        ROUND(count(tweets.id::numeric) /newtable.pop , 7) tweet_percap
    FROM tweets
    JOIN newtable ON ST_Intersects(newtable.geom, tweets.loc)
    GROUP BY newtable.geom, newtable.geoid, newtable.pop;""")

conn.commit()
```

In [32]:
```
conn.close()
```