HW7-postgis (2) (1)

November 5, 2021

[1]:
```
%load_ext sql
%matplotlib inline
```

[32]:
```
import matplotlib.pyplot as plt
import zipfile
import pandas as pd
import geopandas
from sqlalchemy import create_engine
import sqlalchemy.sql
from shapely.geometry import Point, Polygon
```

Nicolas Mavromatis

Nima6629@colorado.edu

1 Problem Set

In this problem set, you're going to help me realize one of my life goals - being able to gaze admiringly on the graceful motions of a windmill while having a cool beer all the while being near my car. Odd though this goal is, the only place I've found it possible to satisfy my odd desire is at the Rocky Flats Lounge in Colorado (since defunct) and now in Manorca, Spain:

Your job is to find where I can live the dream in Colorado. You'll be provided with the following datasets: * A shapefile of the state boundaries from the US Census Bureau * A shapefile of major roadways in the US from the US Census Bureau / TIGER data sources * A map of Wind Turbines in the US from the USGS - note that this data is provided in GeoJSON format. * A collection of the location of bars in the US extracted from Open Street View following a guide on mapping out bierhalls in Germany

Specifically, you will be asked to: * Create PostGIS database tables using the provided information sources * Build PostGIS SQL queries to answer a series of questions * Culuminating in finding a bar, near a road near a windmill.

You'll need to use a PostGIS database and we'll assume you've created one using ElephantSQL

1.1 Load our datasets

[3]:
```
states = geopandas.read_file("zip://./cb_2017_us_state_5m.zip")
windmills=geopandas.read_file('uswtdb_v1_3_20190107.geojson')
roads=geopandas.read_file('zip://./tl_2018_us_primaryroads.zip')
```

Now to the collect the info on bars. This data comes from the OpenStreetMaps project and I used the following query in their Overpass API by adapting a similar query concerning Biergartens in Germany.

```
import requests
import json

overpass_url = "http://overpass-api.de/api/interpreter"
overpass_query = """
[out:json];
area["ISO3166-1"="US"][admin_level=2];
(node["amenity"="bar"](area);
 way["amenity"="bar"](area);
 rel["amenity"="bar"](area);
);
out center;
"""
response = requests.get(overpass_url,
                        params={'data': overpass_query})
data = response.json()
```

This returns a JSON file (provided to you in `bars.json` where bars['elements'] contains information about the ~11,365 bars recorded by OpenStreetMap in the US. In order to treat this data like the windmills and roads, I will construct a GeoPandas GeoDataFrame for you.

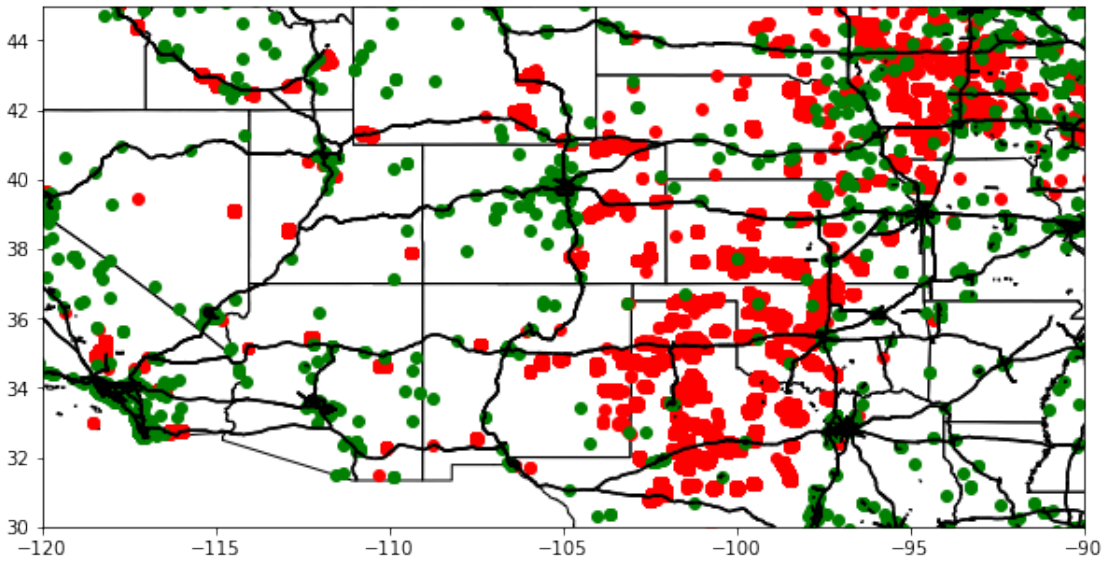
[4]:
```
from shapely.geometry import Point, Polygon, LineString
import json

with open('bars.json') as infile:
    rawbar = json.load(infile)

def getBarName(element):
    if 'name' in element['tags']:
        return element['tags']['name']
    else:
        return 'Unnamed Bar'
barNames = [getBarName(x) for x in rawbar['elements']]
barLat = [x['lat'] for x in rawbar['elements']]
barLon = [x['lon'] for x in rawbar['elements']]

barPoints = [Point(x['lon'], x['lat']) for x in rawbar['elements']]
```

```
bars = geopandas.GeoDataFrame({'name' : barNames, 'lat' : barLat, 'lon' :␣
↪barLon}, geometry=barPoints )
```

We'll just visually inspect the databases by plotting the data. Red is windmills, green is bars and there are roads. If you comment out the `ax.set` line, you can see the full database but it includes roads, bars and windmills in Alaska and Guam, so it's hard to see detail.

[5]:
```
ax = states.plot(color='None', edgecolor='k', figsize=(10,20))
roads.plot(ax = ax, color='black')
windmills.plot(ax = ax, color='red')
bars.plot(ax=ax, color='green')
ax.set(xlim=(-120,-90), ylim=(30,45))
```

[5]:
```
[(-120.0, -90.0), (30.0, 45.0)]
```



1.1.1 Reducing our dataset size

It turns out this is more data than our free database service can handle, so we're going to subset the data to just that available in some western states including Colorado. We could do this by extracting the `geometry` of those states from the `States` table, and then selecting only those bars, windmills and roads that intersect that geometry. However, we're going to define a polygon that captures all of Colorado and just enough of Wyoming,

[6]:
```
## If we wanted to use specific states, you can uncommon this code,
## but the single polygon below captures enough to be useful

#westGeometry = states[ states.NAME=='Colorado' ].geometry.iloc[0]
```

```
#westGeometry = westGeometry.union( states[ states.NAME=='Wyoming' ].geometry.
↪iloc[0])
#westGeometry = westGeometry.union( states[ states.NAME=='New Mexico' ].
↪geometry.iloc[0])

westGeometry = Polygon( [(-110, 42), (-102,42), (-105,35), (-110,35)] )
```

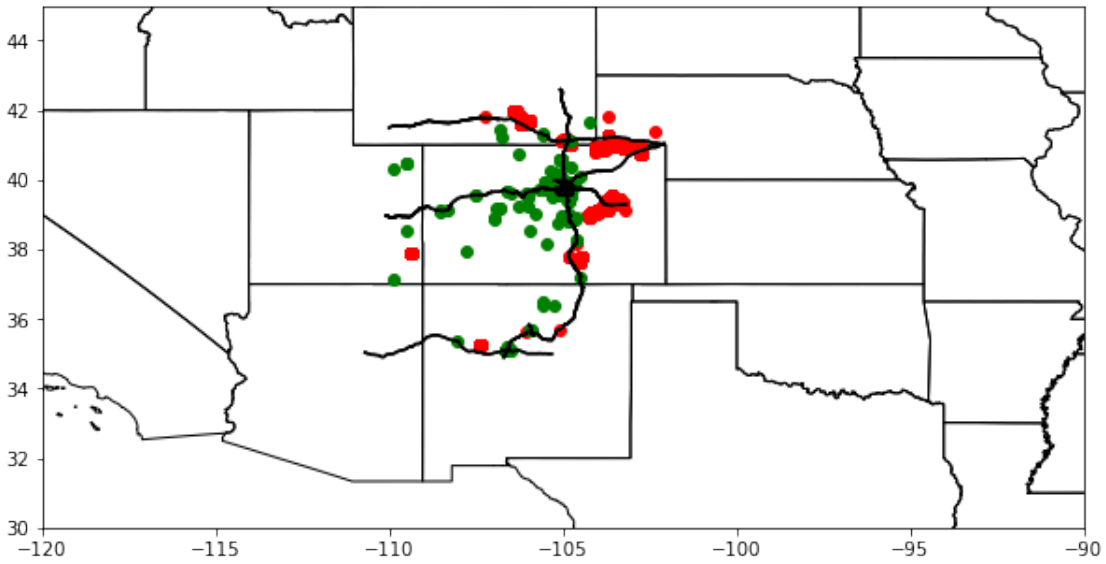
[7]:
```
westRoads = roads[ roads.geometry.intersects( westGeometry) ]
westWindmills = windmills[ windmills.geometry.intersects( westGeometry )]
westBars = bars[ bars.geometry.intersects( westGeometry )]
print("There are {0} roads, {1} windmills and {2} bars".format(len(westRoads),␣
↪len(westWindmills), len(westBars)))
```

```
There are 452 roads, 2356 windmills and 383 bars
```

You'll see that this is much smaller but that some of the roads "stick out" into neighboring states:

[8]:
```
ax = states.plot(color='None', edgecolor='k', figsize=(10,20))
westRoads.plot(ax = ax, color='black')
westWindmills.plot(ax = ax, color='red')
westBars.plot(ax=ax, color='green')
ax.set(xlim=(-120,-90), ylim=(30,45))
```

[8]:
```
[(-120.0, -90.0), (30.0, 45.0)]
```



1.2 Connect to your database & Create Tables

[9]:
```
#import getpass
    # passwd = getpass.getpass()
#dbpasswd = 'YOUR PASSWORD HERE'
#dbuser='YOUR USERNAME HERE'
#dbstr=f"postgresql://{dbuser}:{dbpasswd}@applied-sql.cs.colorado.edu/{dbuser}"
#print("connect to ", dbstr)
#eng = create_engine(dbstr)
#con = eng.connect()
```

[10]:
```
import getpass

#dbstr=f"postgresql://{dbuser}:{dbpasswd}@applied-sql.cs.colorado.edu/{dbuser}"
dbstr = "postgresql://nima6629:9a036fb436fde63ebd5b@applied-sql.cs.colorado.edu/
 ↪nima6629"
try:
    eng = create_engine(dbstr)
    con = eng.connect()
    con.close()
    eng.dispose()
    print("Disconnected from... ", dbstr)
except:
    print("error")
```

```
Disconnected from…  postgresql://nima6629:9a036fb436fde63ebd5b@applied-
sql.cs.colorado.edu/nima6629
```

[11]:
```
import getpass

#dbstr=f"postgresql://{dbuser}:{dbpasswd}@applied-sql.cs.colorado.edu/{dbuser}"
try:
    dbstr = "postgresql://nima6629:9a036fb436fde63ebd5b@applied-sql.cs.colorado.
 ↪edu/nima6629"
    print("connect to ", dbstr)
    eng = create_engine(dbstr)
    con = eng.connect()
except:
    print("CONNECTION ERROR...")
```

```
connect to  postgresql://nima6629:9a036fb436fde63ebd5b@applied-
sql.cs.colorado.edu/nima6629
```

[12]:
```
#Terminate long running processes by running next two cells
```

[13]:
```
con.execute(
'''
```

```
SELECT
pid,
now() - pg_stat_activity.query_start,
query,
state
FROM
pg_stat_activity
WHERE
(now() - pg_stat_activity.query_start) > interval '5 minutes';
'''
).fetchall()
```

[13]:
```
[]
```

[14]:
```
#copy pid into argument below
con.execute(
'''
SELECT pg_cancel_backend(18820);
'''
).fetchall()
```

[14]:
```
[(False,)]
```

We'll run a PostGIS specific query to determine the version of the software available on that postgres server. If this fails with an error, contact the instructor.

[15]:
```
con.execute("SELECT postgis_full_version()").fetchall()
```

[15]:
```
[('POSTGIS="3.1.1 aaf4c79" [EXTENSION] PGSQL="130" GEOS="3.7.1-CAPI-1.11.1
27a5e771" PROJ="Rel. 5.2.0, September 15th, 2018" LIBXML="2.9.4"
LIBJSON="0.12.1" LIBPROTOBUF="1.3.1" WAGYU="0.5.0 (Internal)" TOPOLOGY',)]
```

1.2.1 Check before recreating your data!

It can take a few minutes to load all the data into the database, and you probably want to avoid repeating that over and over.

So, in the following code

[16]:
```
try:
    roadsInDb = con.execute('SELECT count(*) from roads;').fetchall()
    print("You currently have {0} roads defined of {1} possible".format(␣
 ↪roadsInDb, len(westRoads)) )

    statesInDb = con.execute('SELECT count(*) from states;').fetchall()
    print("You currently have {0} states defined of {1} possible".format(␣
 ↪statesInDb, len(states)) )
```

```
    windmillsInDb = con.execute('SELECT count(*) from windmills;').fetchall()
    print("You currently have {0} windmills defined of {1} possible".format(␣
 ↪windmillsInDb, len(westWindmills)) )

    barsInDb = con.execute('SELECT count(*) from bars;').fetchall()
    print("You currently have {0} bars defined of {1} possible".format(␣
 ↪barsInDb, len(westBars) ) )
except:
    print("Looks like you're missing one or more tables")
```

```
You currently have [(452,)] roads defined of 452 possible
You currently have [(56,)] states defined of 56 possible
You currently have [(2356,)] windmills defined of 2356 possible
You currently have [(383,)] bars defined of 383 possible
```

[17]:
```
con.execute("DROP TABLE IF EXISTS hoods;")
```

[17]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x7f8054551e80>
```

1.2.2 Create and populate your roads table [5 pts]

You should define the name of the road and the location (geography) and populate it using the subset of roads, `bigroads`. Note that road names may not been unique (check the data). You should structure this as one cell to create the table and another cell to populate the table. Because it takes a while to populate the table, you'll want to be able to selectively execute these statements.

[38]:
```
con.execute('''
DROP TABLE IF EXISTS roads;
CREATE TABLE roads
    (
      name VARCHAR(1000),
      geom GEOGRAPHY
    );
''')
```

[38]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x7f4c7c6a2460>
```

[39]:
```
%%time
for road in westRoads.to_dict('records'):
    road['wkt'] = road['geometry'].wkt
    cmd = sqlalchemy.sql.text('''INSERT INTO roads(name, geom)
            VALUES ( :FULLNAME, ST_SetSRID(ST_GeogFromText(:wkt), 4326) )''')
    con.execute(cmd, road)
```

```
CPU times: user 710 ms, sys: 97.6 ms, total: 808 ms
Wall time: 2.11 s
```

1.2.3 Create and populate your states table [5 pts]

You should define the name of the state and the location (geography).

[40]:
```
con.execute('''
DROP TABLE IF EXISTS states;
CREATE TABLE states
    (
        name VARCHAR(1000),
        geom GEOGRAPHY
    );
''')
```

[40]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x7f4c7c8d07c0>
```

[41]:
```
%%time
cmd = sqlalchemy.sql.text('''INSERT INTO states(name, geom)
                                    VALUES ( :STUSPS, ST_SetSRID(ST_GeogFromText(:
 ↪wkt), 4326)) ''')

for st in states.to_dict('records'):
    st['wkt'] = st['geometry'].wkt
    con.execute(cmd, st)
```

```
CPU times: user 379 ms, sys: 12.8 ms, total: 392 ms
Wall time: 758 ms
```

1.2.4 Create and populate your `windmills` table [5 pts]

You just need to define the location (geography) of the windmill. You should define a `case_id` field so you can tell the windmills apart and define a `GEOGRAPHY` for the location of the windmill.

[42]:
```
con.execute('''
DROP TABLE IF EXISTS windmills;
CREATE TABLE windmills
(
     id integer,
     geom GEOGRAPHY
);
''')
```

[42]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x7f4c7c8e56a0>
```

[43]:
```
%%time
for mill in westWindmills.to_dict('records'):
    cmd = sqlalchemy.sql.text('''INSERT INTO windmills(id, geom)
                                      VALUES ( :case_id, ST_SetSRID(ST_MakePoint(:
 ↪xlong, :ylat), 4326))''')
    con.execute(cmd, mill)
```

```
CPU times: user 891 ms, sys: 280 ms, total: 1.17 s
Wall time: 6.31 s
```

1.2.5 Create and populate your bars table [5 pts]

You need to define the name, lat, lon and location (geography) of the bars.

[44]:
```
con.execute('''
DROP TABLE IF EXISTS bars;
CREATE TABLE bars
        (
      name varchar(1000),
      lat numeric,
      lon numeric,
      geom GEOGRAPHY
        );
''')
```

[44]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x7f4c7c69b220>
```

[45]:
```
%%time
for bar in westBars.to_dict('records'):
    bar['wkt'] = bar['geometry'].wkt
    cmd = sqlalchemy.sql.text('''INSERT INTO bars(name, lat, lon, geom)
                                 VALUES ( :name, :lat, :lon,␣
 ↪ST_SetSRID(ST_GeogFromText(:wkt), 4326 ))''')
    con.execute(cmd, bar)
```

```
CPU times: user 203 ms, sys: 67.4 ms, total: 271 ms
Wall time: 1.03 s
```

1.3 The Queries

From here on out, you will use the data in the PostGIS database to answer the following questions.

1.3.1 Query #1: Find the distance from a bar to a windmill [10 pts]

Print the name of bars and the ID of windmills and the distance (in meters) between the bar and the windmill order by distance. You should only print the first 10 entries. It should start with

```
[('Brick House Tavern', 3010715, 2305.36102164),
 ('Danielmark Brewing Co.', 3015904, 6997.97151555),
 ('Danielmark Brewing Co.', 3015903, 7358.48396178),
```

[22]:
```
%%time
#ST_DISTANCE(Geography1, geography2) calcs distance in geo points...geom is␣
 ↪geog. point! It is misnamed.
con.execute('''
    SELECT b.name, w.ID,  ST_Distance(w.geom, b.geom) as d
```

```
    FROM bars AS b, windmills AS w
    ORDER BY d ASC
    LIMIT 10;
''').fetchall()
```

```
CPU times: user 4 µs, sys: 4.05 ms, total: 4.06 ms
Wall time: 2.28 s
```

[22]:
```
[('Brick House Tavern', 3010715, 2305.36102164),
 ('Danielmark Brewing Co.', 3015904, 6997.97151555),
 ('Danielmark Brewing Co.', 3015903, 7358.48396178),
 ('Scooters Sports Bar & Grill', 3010715, 7385.13380769),
 ('Club On Tooy', 3010715, 7606.01028836),
 ('Danielmark Brewing Co.', 3023379, 7628.35525503),
 ('Miles Sports Pub & Grub', 3072850, 7756.22598172),
 ('Oasis Bar & Grill', 3010715, 7970.13862602),
 ('Miles Sports Pub & Grub', 3002673, 8166.8534879),
 ('Miles Sports Pub & Grub', 3004706, 8302.26821158)]
```

1.3.2 Query #2: Find the distance to the *nearest* windmill from each bar [10 pts]

For each bar, determine the distance to the nearest windmill. You should print the bar name and the distance to the closest windmill. You should only print the first 10 entries. It should start with

```
[('Brick House Tavern', 2305.36102164),
 ('Danielmark Brewing Co.', 6997.97151555),
```

[23]:
```
%%time
#Group by bar name, then find min distance between any windmill and that bar␣
 ↪group
con.execute('''
    SELECT b.name, MIN(ST_Distance(w.geom, b.geom)) as d
    FROM bars AS b, windmills AS w
    GROUP BY b.name
    ORDER BY d ASC
    LIMIT 10;
''').fetchall()
```

```
CPU times: user 3.09 ms, sys: 187 µs, total: 3.28 ms
Wall time: 2.8 s
```

[23]:
```
[('Brick House Tavern', 2305.36102164),
 ('Danielmark Brewing Co.', 6997.97151555),
 ('Scooters Sports Bar & Grill', 7385.13380769),
 ('Club On Tooy', 7606.01028836),
 ('Miles Sports Pub & Grub', 7756.22598172),
 ('Oasis Bar & Grill', 7970.13862602),
 ('Pool Bar', 8820.72453063),
```

```
 ('Si Amigos', 9242.36551446),
 ('Outlaw Saloon', 9275.24910324),
 ('La Colonia Bar', 9582.59905431)]
```

1.3.3 Query #3: Find the distance to the nearest windmill but also tell me what state the bar is in [10 pts]

As before, for each bar, determine the distance to the nearest windmill. You should print the bar name, **the state the bar is in**, and the distance to the closest windmill. You should only print the first 10 entries. It should start with

```
[('Brick House Tavern', 'CO', 2305.36102164),
 ('Danielmark Brewing Co.', 'WY', 6997.97151555),
```

[25]:
```
%%time
#HERE we have to cast a geography type to geometry type to work with ST_WITHIN
#As in 'b.geom::geometry'
#We also do the where condition to make sure that the bar is within the state,
#grouping by bar, state
con.execute('''
    SELECT b.name, s.name, MIN(ST_Distance(w.geom, b.geom)) AS d
    FROM bars AS b, windmills AS w, states AS s
    WHERE ST_WITHIN(b.geom::geometry, s.geom::geometry)
    GROUP BY b.name, s.name
    ORDER BY d ASC
    LIMIT 10;
''').fetchall()
```

```
CPU times: user 0 ns, sys: 3.04 ms, total: 3.04 ms
Wall time: 3.16 s
```

[25]:
```
[('Brick House Tavern', 'CO', 2305.36102164),
 ('Danielmark Brewing Co.', 'WY', 6997.97151555),
 ('Scooters Sports Bar & Grill', 'CO', 7385.13380769),
 ('Club On Tooy', 'CO', 7606.01028836),
 ('Miles Sports Pub & Grub', 'CO', 7756.22598172),
 ('Oasis Bar & Grill', 'CO', 7970.13862602),
 ('Pool Bar', 'CO', 8820.72453063),
 ('Si Amigos', 'CO', 9242.36551446),
 ('Outlaw Saloon', 'WY', 9275.24910324),
 ('La Colonia Bar', 'CO', 9582.59905431)]
```

1.3.4 Query #4: Show me bars in Colorado that are within 10km of a windmill using `ST_Distance` [10 pts]

Pretty much like it says – provide a list of bars that are within 10km of a windmill. Your code **must** use the `ST_Distance` spatial query.

Each bar should be listed **once** even if there are two windmills within 10km and only the distance to the closest windmill should be displayed. You should print all entries (there are 10).

It should start with

```
[('Brick House Tavern', 'CO', 2305.36102164),
 ('Danielmark Brewing Co.', 'WY', 6997.97151555),
```

You should note the "wall time" needed for this query using the `%%time` magic.

[26]:
```
%%time
#TASK: Find the min distance (must be<=10000 units) from each bar, within CO␣
 ↪and WY, to a windmill.

#Select the bar name, min distance from windmill to bar, ALIAS minD, and the␣
 ↪state
#where the distance between the windmill and bar is <=10000 units,
#AND the bar and state geometry MUST intersect (states only have WY, CO),
#Finally group by bar.name, state.name, order by minD.

con.execute('''
    SELECT
    bars.name,
    states.name,
    min(ST_Distance(windmills.geom, bars.geom)) AS minD
    FROM bars, windmills, states
    WHERE ST_Distance(windmills.geom, bars.geom) <= 10000
    AND
    ST_Intersects(bars.geom, states.geom)
    GROUP BY bars.name, states.name
    ORDER BY minD ASC
    LIMIT 10;
''').fetchall()
```

```
CPU times: user 4.42 ms, sys: 578 µs, total: 4.99 ms
Wall time: 22.2 s
```

[26]:
```
[('Brick House Tavern', 'CO', 2305.36102164),
 ('Danielmark Brewing Co.', 'WY', 6997.97151555),
 ('Scooters Sports Bar & Grill', 'CO', 7385.13380769),
 ('Club On Tooy', 'CO', 7606.01028836),
 ('Miles Sports Pub & Grub', 'CO', 7756.22598172),
 ('Oasis Bar & Grill', 'CO', 7970.13862602),
 ('Pool Bar', 'CO', 8820.72453063),
 ('Si Amigos', 'CO', 9242.36551446),
 ('Outlaw Saloon', 'WY', 9275.24910324),
 ('La Colonia Bar', 'CO', 9582.59905431)]
```

1.3.5 Query #5: Show me bars in Colorado that are within 10km of a windmill using `ST_Buffer` [10 pts]

This is the same as the previous query but this time, you must use the `ST_Buffer` method.

Each bar should be listed **once** even if there are two windmills within 10km. You should only print the first 10 entries. It should start with

```
[('Brick House Tavern', 'CO', 2305.36102164),
 ('Danielmark Brewing Co.', 'WY', 6997.97151555),
```

[76]:
```
%%time
#TASK: Find the min distance (must be<=10000 units) from each bar, within CO␣
 ↪and WY, to a windmill.

#Select the bar name, min distance from windmill to bar, ALIAS minD
#where the distance between the windmill and bar is <=10000 units,
#BY checking the itersection between two geom points of radius 10000/2, which␣
 ↪is equivalent.
#in code: ST_intersects(ST_Buffer(bars.geom::geometry, 10000/2),␣
 ↪ST_Buffer(windmills.geom::geometry, 10000/2))
#INNER SFW alias w,
#Outer loop: select attrs where the minD bar.name is within the state geometry.
 ↪(states only have WY, CO)

#This runs much faster with a NSFW to reduce number of tuples to iterate over.
#ALSO:  ST_contains(s.geom::geometry, b.geom::geometry) runs much faster on␣
 ↪geom points
#than ST_Intersects(s.geom, b.geom) on geog points (slow!)

con.execute('''
    SELECT
    b.name,
    s.name,
    w.minD
    FROM
    (
        SELECT
        bars.name as bName,
        min(ST_Distance(windmills.geom, bars.geom)) AS minD
        FROM
        bars, windmills
        WHERE
        ST_intersects(ST_Buffer(bars.geom::geometry, 5000), ST_Buffer(windmills.
 ↪geom::geometry, 5000))
        GROUP BY bName
    ) AS w,
    bars AS b, states AS s
```

```
    WHERE
    w.bName = b.name
    AND
    ST_contains(s.geom::geometry, b.geom::geometry)
    ORDER BY w.minD
    LIMIT 10;
''').fetchall()
```

```
CPU times: user 4.67 ms, sys: 596 µs, total: 5.26 ms
Wall time: 43.8 s
```

[76]:
```
[('Brick House Tavern', 'CO', 2305.36102164),
 ('Danielmark Brewing Co.', 'WY', 6997.97151555),
 ('Scooters Sports Bar & Grill', 'CO', 7385.13380769),
 ('Club On Tooy', 'CO', 7606.01028836),
 ('Miles Sports Pub & Grub', 'CO', 7756.22598172),
 ('Oasis Bar & Grill', 'CO', 7970.13862602),
 ('Pool Bar', 'CO', 8820.72453063),
 ('Si Amigos', 'CO', 9242.36551446),
 ('Outlaw Saloon', 'WY', 9275.24910324),
 ('La Colonia Bar', 'CO', 9582.59905431)]
```

1.3.6 Query #6: Measure the impact of Geospatial Indexes [10 pts]

Queries #4 and #5 measured the time to find the distance from each bar to the nearest windmill that is less than 10km away using two different methods. You should have found that one method is much faster than the other using the "Wall time" output of the `%%time` magic.

You are now going to create an index for the geometries of the bars and windmills. and then measure the execution time of the queries. You should fill in the table in the markdown box below. You should probably structure your index creation in two parts – in the first, you should `DROP INDEX IF..` to allow you to remove the index if needed and follow that by code that `CREATE INDEX ... ON` to create the specific indexes.

Once you've created your indexes, you should rerun queries #4 and #5 and fill in the query times for with/with-out indexes in the table below. You will then be asked to explain WHY the queries have different execution times. It may be useful to use `EXPLAIN SELECT...` to draw your conclsusions.

[67]:
```
%%time
con.execute('''
DROP INDEX IF EXISTS bars_gist;
DROP INDEX IF EXISTS windmills_gist;
DROP INDEX IF EXISTS states_gist;

DROP INDEX IF EXISTS bars_name;
DROP INDEX IF EXISTS windmills_name;
DROP INDEX IF EXISTS states_name;
''')
```

```
CPU times: user 4.39 ms, sys: 106 µs, total: 4.5 ms
Wall time: 11 ms
```

[67]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x7f4c7c9879d0>
```

[68]:
```
%%time
#using GIST(geom) is necessary to create an idx on special geog type.
con.execute('''
CREATE INDEX bars_gist ON bars using GIST(geom);
CREATE INDEX windmills_gist ON windmills using GIST(geom);
CREATE INDEX states_gist ON states using GIST(geom);

CREATE INDEX bars_name ON bars(name);
CREATE INDEX windmills_name ON windmills(ID);
CREATE INDEX states_name ON states(name);
''')
```

```
CPU times: user 19 µs, sys: 4.06 ms, total: 4.08 ms
Wall time: 27.9 ms
```

[68]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x7f4c7c987790>
```

[72]:
```
%%time
#IN order to actually activate the indices, ANALYZE _idx must be called
con.execute('''
ANALYZE bars_gist;
ANALYZE windmills_gist;
ANALYZE states_gist;
ANALYZE bars_name;
ANALYZE windmills_name;
ANALYZE states_name;
''')
```

```
CPU times: user 2.67 ms, sys: 0 ns, total: 2.67 ms
Wall time: 2.31 ms
```

[72]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x7f4c7c8e5220>
```

[73]:
```
%%time
#Q4 once again
con.execute('''
    SELECT
    bars.name,
    states.name,
    min(ST_Distance(windmills.geom, bars.geom)) AS minD
    FROM bars, windmills, states
    WHERE ST_Distance(windmills.geom, bars.geom) <= 10000
    AND
```

```
    ST_Intersects(bars.geom, states.geom)
    GROUP BY bars.name, states.name
    ORDER BY minD ASC
    LIMIT 10;
''').fetchall()
```

```
CPU times: user 533 µs, sys: 3.61 ms, total: 4.15 ms
Wall time: 1.96 s
```

[73]:
```
[('Brick House Tavern', 'CO', 2305.36102164),
 ('Danielmark Brewing Co.', 'WY', 6997.97151555),
 ('Scooters Sports Bar & Grill', 'CO', 7385.13380769),
 ('Club On Tooy', 'CO', 7606.01028836),
 ('Miles Sports Pub & Grub', 'CO', 7756.22598172),
 ('Oasis Bar & Grill', 'CO', 7970.13862602),
 ('Pool Bar', 'CO', 8820.72453063),
 ('Si Amigos', 'CO', 9242.36551446),
 ('Outlaw Saloon', 'WY', 9275.24910324),
 ('La Colonia Bar', 'CO', 9582.59905431)]
```

[75]:
```
%%time
#Q5 once again
con.execute('''
    SELECT
    b.name,
    s.name,
    w.minD
    FROM
    (
        SELECT
        bars.name as bName,
        min(ST_Distance(windmills.geom, bars.geom)) AS minD
        FROM
        bars, windmills
        WHERE
        ST_intersects(ST_Buffer(bars.geom::geometry, 5000), ST_Buffer(windmills.
 ↪geom::geometry, 5000))
        GROUP BY bName
    ) AS w,
    bars AS b, states AS s
    WHERE
    w.bName = b.name
    AND
    ST_contains(s.geom::geometry, b.geom::geometry)
    ORDER BY w.minD
    LIMIT 10;
''').fetchall()
```

```
CPU times: user 1.79 ms, sys: 4.29 ms, total: 6.08 ms
Wall time: 43.6 s
```

[75]:
```
[('Brick House Tavern', 'CO', 2305.36102164),
 ('Danielmark Brewing Co.', 'WY', 6997.97151555),
 ('Scooters Sports Bar & Grill', 'CO', 7385.13380769),
 ('Club On Tooy', 'CO', 7606.01028836),
 ('Miles Sports Pub & Grub', 'CO', 7756.22598172),
 ('Oasis Bar & Grill', 'CO', 7970.13862602),
 ('Pool Bar', 'CO', 8820.72453063),
 ('Si Amigos', 'CO', 9242.36551446),
 ('Outlaw Saloon', 'WY', 9275.24910324),
 ('La Colonia Bar', 'CO', 9582.59905431)]
```

Fill in the table and explain why you got these results:

USING TOTAL CPU (Not wall) TIME

TABLE did not print well, so I did it in text format.

```
ST_Distance: NO: 4.99 ms, YES: 4.15 ms
ST_Buffer:   NO: 5.26 ms, YES: 6.08 ms
```

Explanation: "Given in assignment:The distance calculation on the surface can't make use of the index, but the `ST_Buffer` method reduces to an index search.".
Here, it looks like the times recorded are minimally different, so the indices don't speed up the queries significantly for this implementation.

1.3.7 Query #7: Find the distance between each bar in Colorado and the nearest road [10 pts]

Each bar should be listed **once** with the distance to the nearest road, ordered by the shortest distance in meters between bar and road. You should only print the first 10 entries. It should start with

```
[('Hideaway Bar & Grill', 'CO', 43.7382199),
 ('Pixie Inn', 'CO', 47.06407311),
```

[96]:
```
%%time
#The important thing here, as before, is
#To keep track of 2 instances of barName (aliases)
#The inner SFW selects the bName, minDistance
#from each distinct bar to road (group by bName)
#The outer SFW correlates a specific intersecting state ('CO')
#doing a join on bar name.
#Trying to do all this in one SFW is way slower...
#Goes to show that reducing num of tuples in NSFW is efficient.
con.execute('''
    SELECT
```

```
    bars.name,
    states.name,
    minD
    FROM
    (
    SELECT DISTINCT bars.name as bName,
    min(ST_Distance(roads.geom, bars.geom)) AS minD
    FROM
    bars,
    roads
    GROUP BY bName
    ) AS r,
    states,
    bars
    WHERE
    states.name = 'CO'
    AND
    bars.name=bName
    AND
    ST_Intersects(bars.geom, states.geom)
    ORDER BY minD ASC
    LIMIT 10
''').fetchall()
```

```
CPU times: user 0 ns, sys: 4.98 ms, total: 4.98 ms
Wall time: 26.4 s
```

[96]:
```
[('Hideaway Bar & Grill', 'CO', 43.7382199),
 ('Pixie Inn', 'CO', 47.06407311),
 ("J D's Neighborhood Bar", 'CO', 49.80980978),
 ('Ste. Ellie', 'CO', 67.89244671),
 ('Carbon Cafe & Bar', 'CO', 75.56797075),
 ('Westwinds Tavern', 'CO', 77.38517713),
 ("Stranahan's Lounge", 'CO', 79.67593138),
 ('The Dirty Duck Bar', 'CO', 111.69183711),
 ('Outer Range Brewing Company', 'CO', 133.89596788),
 ("Em's Velvet Vine Wine Bar", 'CO', 136.18861189)]
```

1.3.8 Query #8: Fulfill my life long dream [10 pts]

We now seek to find the bar in Colorado that minimizes the sum of the distance between the road and the nearest windmill. However, the windmill must be within 10km so that I can see it.

There should be exactly one solution to this query. It is very likely you'll need to make use of `with` clauses. The actual bar has relocated and a new bar has openned in its place, so if you look up the lat/lon on the map, you'll find a restaurant but not one with the same name.

You should print out the bar name, the lat, lon, windmill identifier, the name of the closest road

and the sum of the distance to the road and the windmill.

[31]:
```
%%time
#Select minSum as the distance between each bar and road + each bar and␣
 ↪windmill.
#The windmill distance must be <=10000, in CO, and within the state
#SELECT the MIN of minSum.distance
#joininh on distance, and road.name

#The key is to define aliases for entire SFW clauses to deliniate instances of␣
 ↪distance
con.execute('''
    WITH
    msum AS (
            SELECT
            DISTINCT bars.name as barname,
            windmills.id as wm,
            roads.name as rd,
            ST_Distance(bars.geom, roads.geom)+ST_Distance(bars.geom, windmills.
 ↪geom) as d
            from
            bars, windmills, roads, states
            WHERE
            states.name='CO'
            AND
            ST_Intersects(bars.geom, states.geom)
            AND
            ST_Distance(bars.geom, windmills.geom) <= 10000
            ),
        minD AS (
        SELECT MIN(msum.d) as d
        FROM msum )
    SELECT msum.barname, bars.lat, bars.lon, msum.wm, roads.name, msum.d
    FROM minD, msum, roads, bars
    WHERE msum.d = minD.d
    AND msum.rd = roads.name
    AND bars.name = msum.barname
    LIMIT 1;
    ''').fetchall();
```

```
CPU times: user 4.39 ms, sys: 0 ns, total: 4.39 ms
Wall time: 2.97 s
```

[31]:
```
[('Brick House Tavern', Decimal('39.7815195'), Decimal('-104.7714205'), 3010715,
 'Pena Blvd', 3940.4826017800006)]
```