Using PostGIS in a real advanced way !

FOSS4G 2016 – Bonn

Try to enlarge your vision
about what is possible to do with PostgreSQL/PostGIS

#1 About PostgreSQL

PostgreSQL: modern SQL

Keep enhancing and improving features

- CTE and Recursive CTE
- Windowing
- Aggregate Function
- Lateral
- Array
- Table Inheritance

LATERAL and KNN

```
SELECT
   bus.gid, bus.nom, lat.gid, lat.nom, lat.dist
FROM
   own.tcl as bus
   , LATERAL (
      SELECT
         bar.gid, bar.nom, bar.geom
         , ST_Distance(bar.geom, bus.geom) as dist
      FROM
         own.water as bar
      ORDER BY
         bar.geom <-> bus.geom   -- forbidden without lateral
      LIMIT 2
   ) AS lat
ORDER BY
   bus.gid, lat.dist desc;
```

#1 About PostgreSQL

Since PostgreSQL 9.1 : EXTENSION handling

Using existing extension is that easy, UUID generation example :

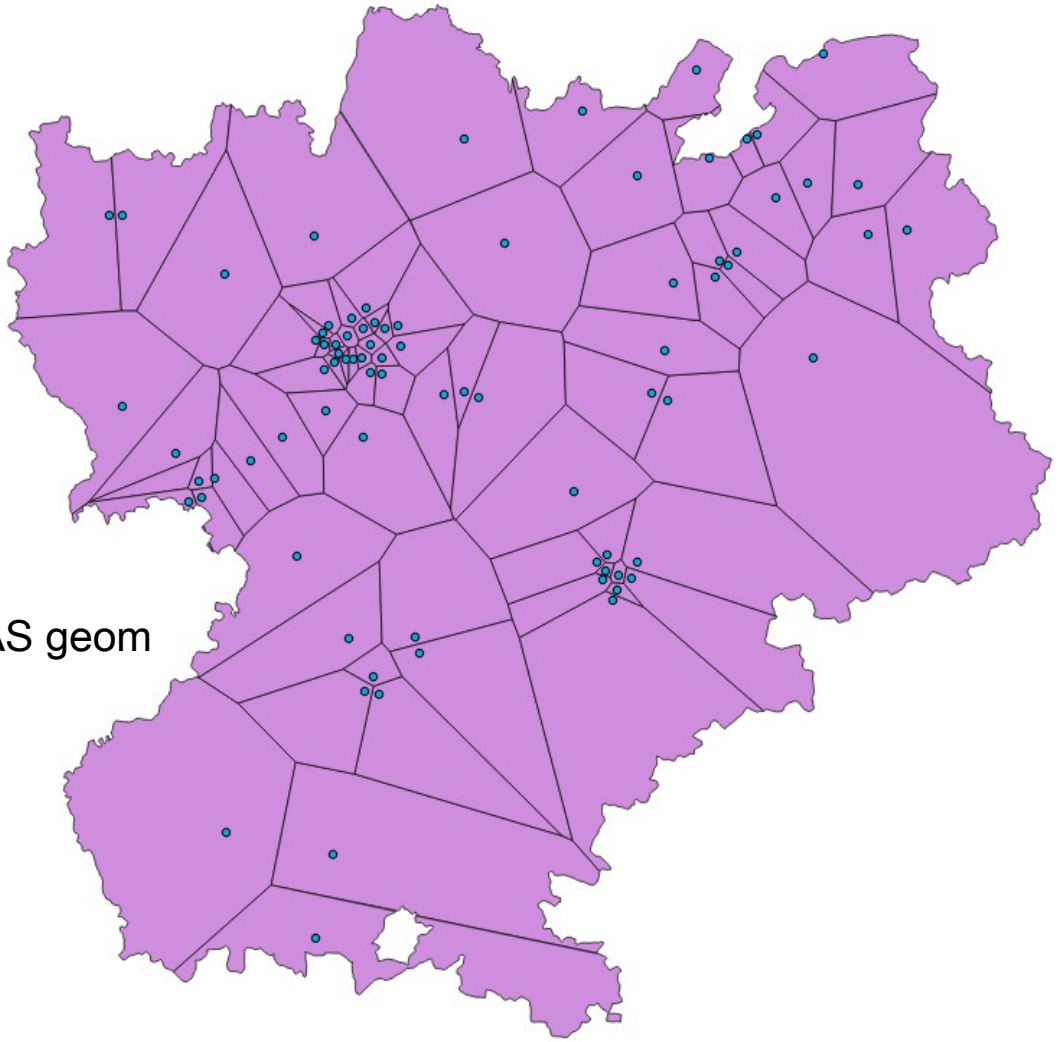
```
foo=# CREATE EXTENSION "uuid-ossp";
CREATE EXTENSION

foo=# SELECT uuid_generate_v4();

6953879c-3aae-4d42-a470-6d430305e173
```

Pl/Python basic Geocoder function

```
CREATE OR REPLACE FUNCTION geoname(toponym text)
                    RETURNS geometry(Point,4326)
AS $$

    from geopy import geocoders
    g = geocoders.GeoNames(username="YOUR_USERNAME")

    try:
        place, (lat, lng) = g.geocode(toponym)

        result = plpy.execute(
        "SELECT 'SRID=4326;POINT(%s %s)'::geometry(Point, 4326) AS geom"
        % (lng, lat), 1)

        return result[0]["geom"]

    except:
        plpy.warning('Geocoding Error')
        return None


$$ LANGUAGE plpython3u;
```

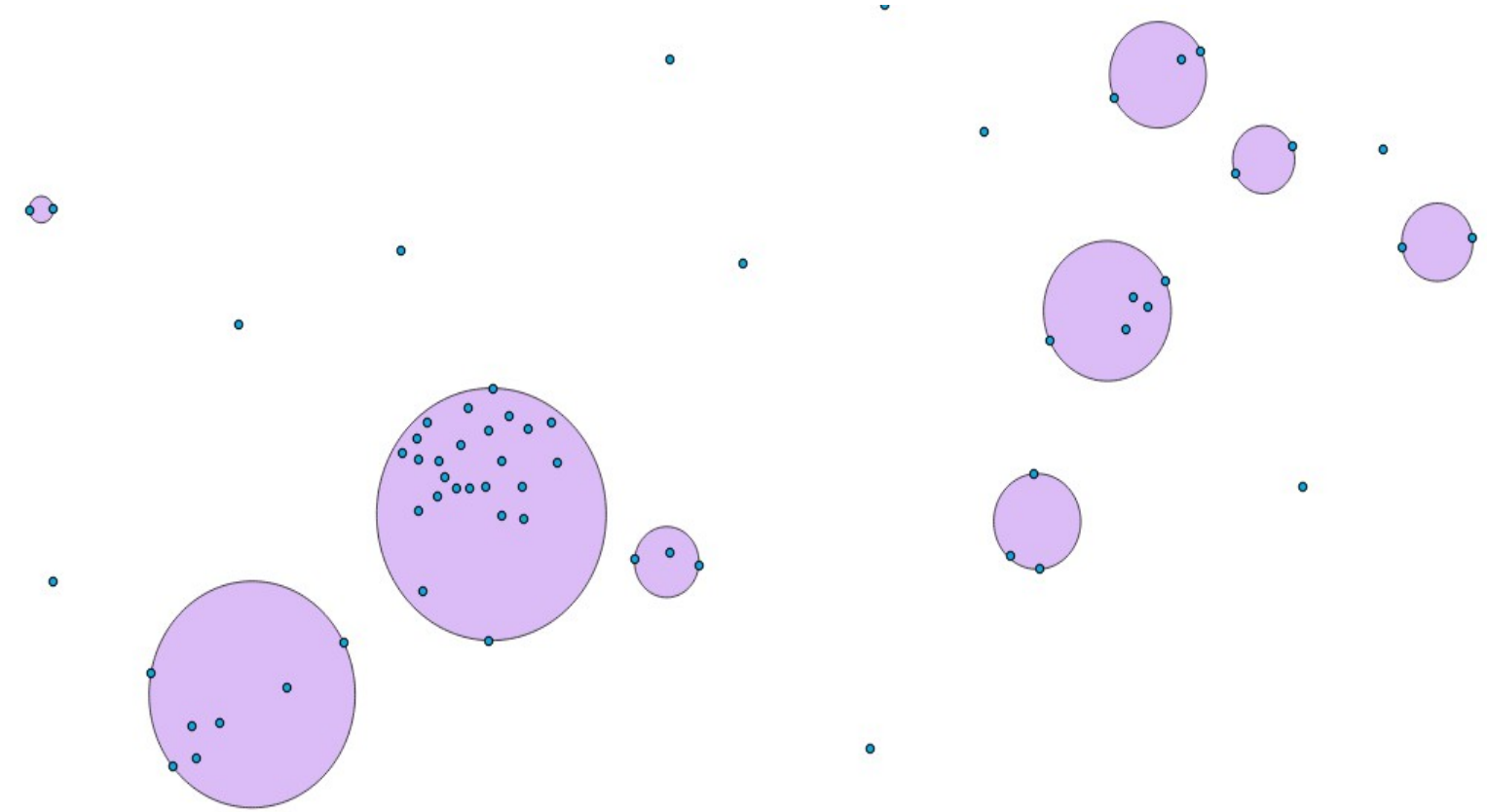
```
psql db -c
"SELECT ST_AsGeoJSON(geoname('New York, NY 10022'))"

{"type":"Point","coordinates":[-74.00597,40.71427]}
```

ST_Voronoi



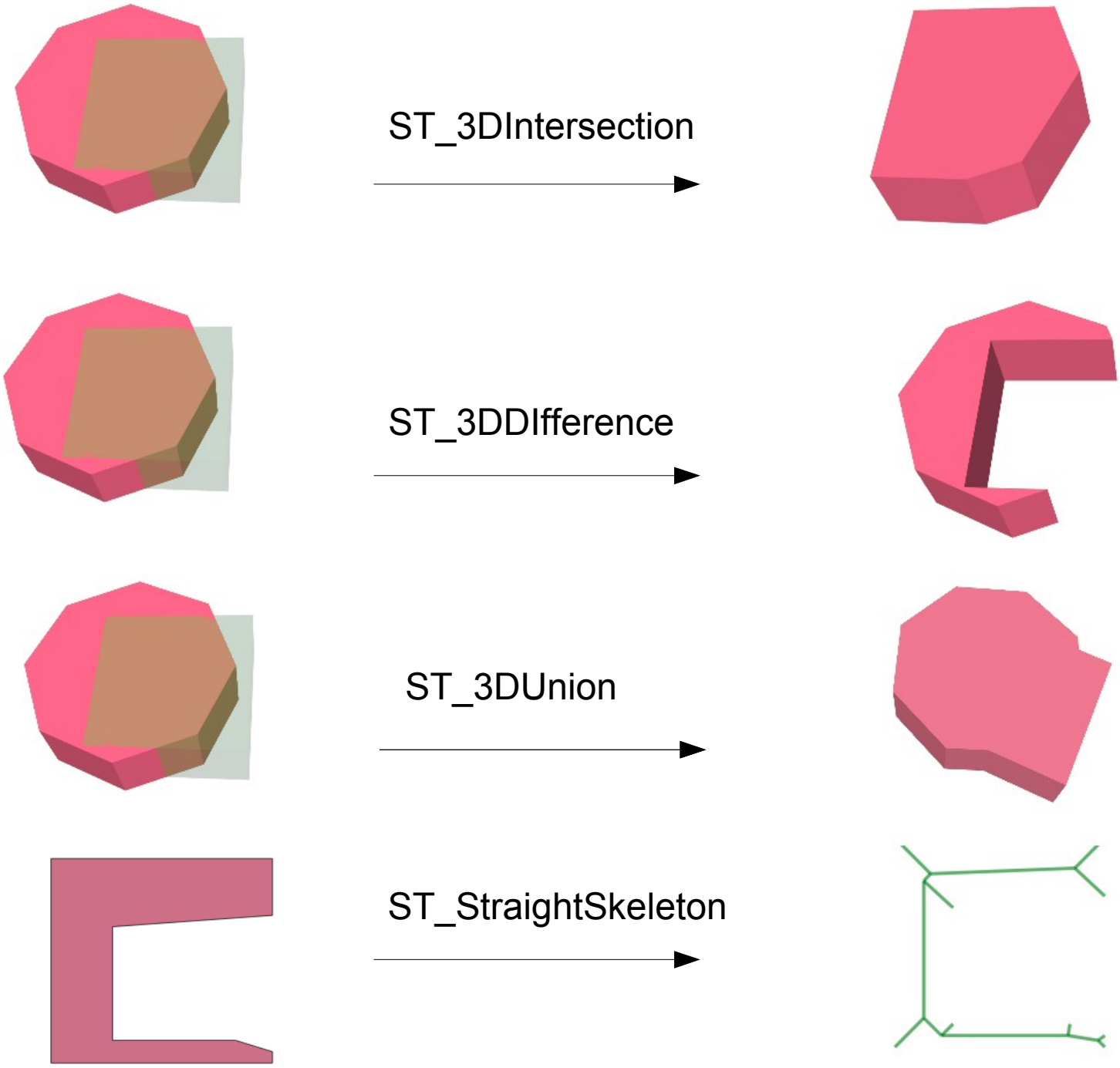
```
WITH c AS (

  SELECT ST_Centroid((ST_Dump(geom)).geom) AS geom
  FROM own.commune
  WHERE population > 10000

), v AS (

  SELECT ST_Intersection (

      (ST_Dump(ST_CollectionHomogenize(ST_Voronoi(ST_Collect(geom))))).geom,
            (SELECT ST_Union(geom) FROM own.commune)
        ) AS geom
  FROM c
)

SELECT geom, row_number() OVER() AS id FROM v
```

ST_ClusterWithin



```
WITH

 c AS (
   SELECT unnest(ST_ClusterWithin(ST_Centroid(geom), 12000)) AS geom
   FROM own.commune
   WHERE population > 10000
 )

 SELECT row_number() OVER() AS id, ST_MinimumBoundingCircle(geom) AS geom
 FROM c
 WHERE ST_NumGeometries(geom) > 1
```

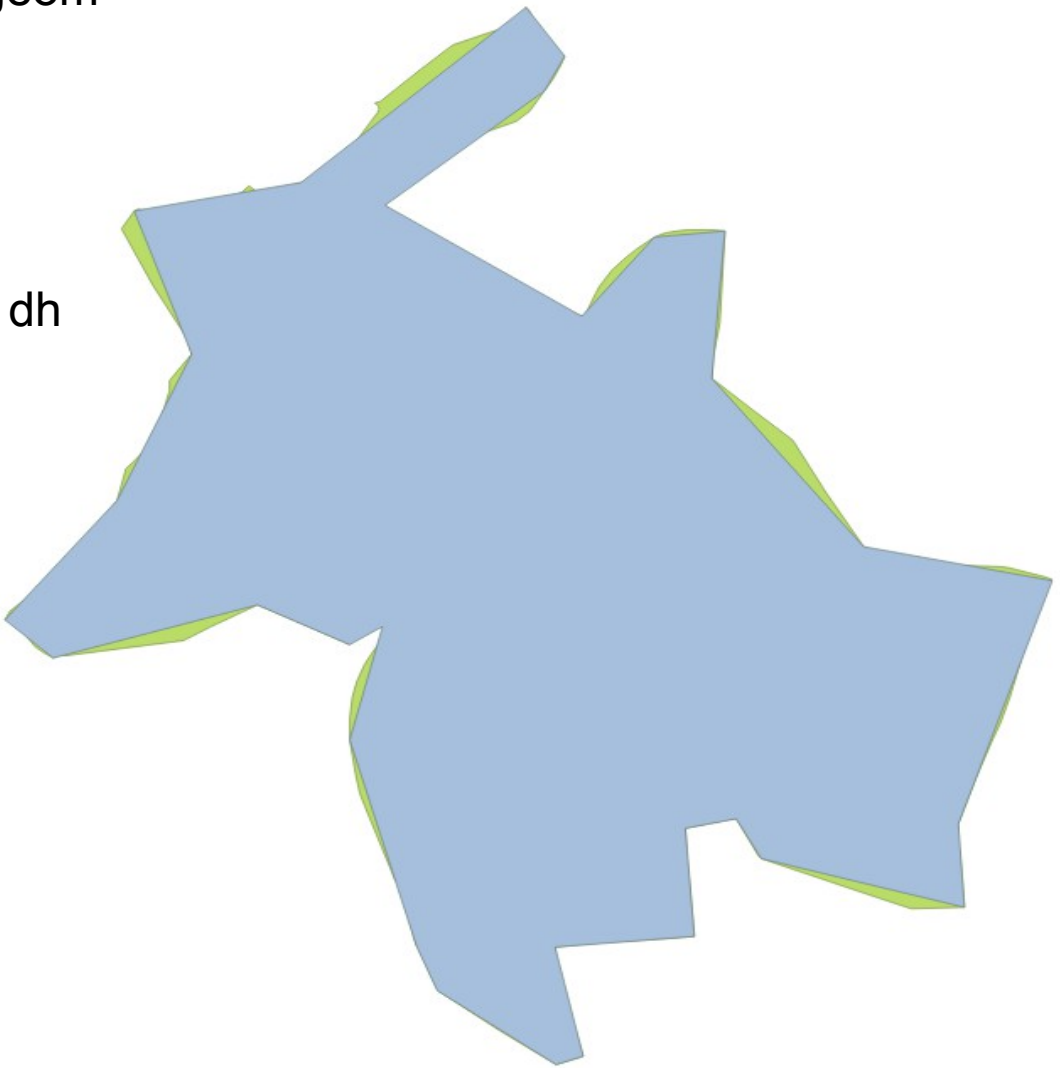
SFCGAL Extension



```
CREATE EXTENSION fuzzystrmatch;



SELECT levenshtein ('same', 'same');    - - and not different
0



SELECT levenshtein ('gdal', 'pdal');
1



SELECT levenshtein ('postgis', 'oracle spatial');
12
```

ST_HausdorffDistance

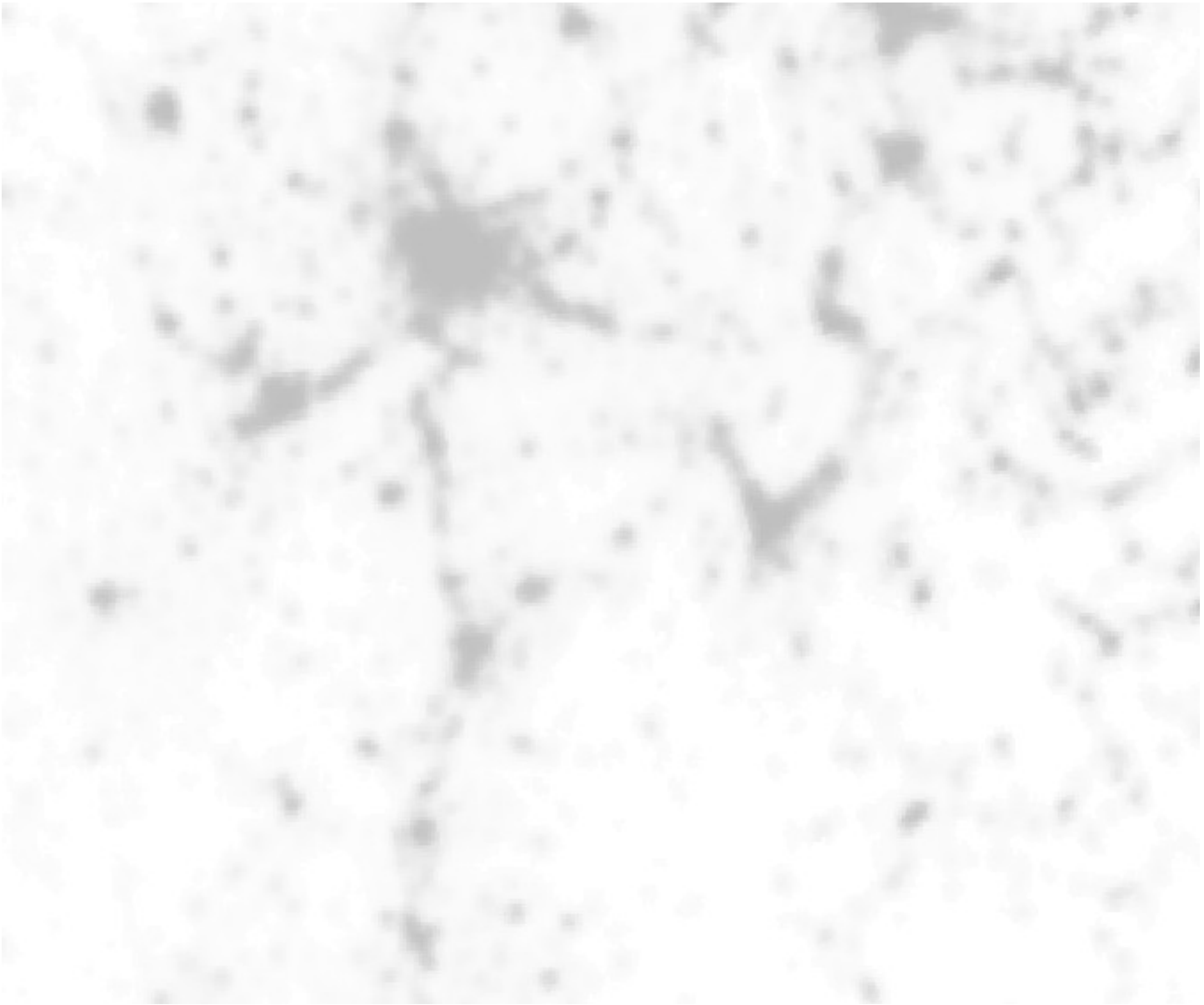
```
WITH a AS (
  SELECT id, ST_Simplify(geom, 5000) AS geom
  FROM own.commune
)

SELECT a.id, b.id,
ST_HausdorffDistance(a.geom, b.geom) AS dh
FROM a, own.commune b
WHERE nom_com = 'Lyon'
ORDER BY dh ASC
LIMIT 5;
```



```
 id  |  id  |        dh
------+------+------------------
 1347 | 1347 | 185.139093997864
 1072 | 1347 | 6681.60493070321
 2461 | 1347 | 6817.89817025694
 2824 | 1347 | 7149.21791806655
  344 | 1347 | 7929.70883765602
```

But, could we get a bit deeper in our (spatial) analysis ?

Light Pollution @Night

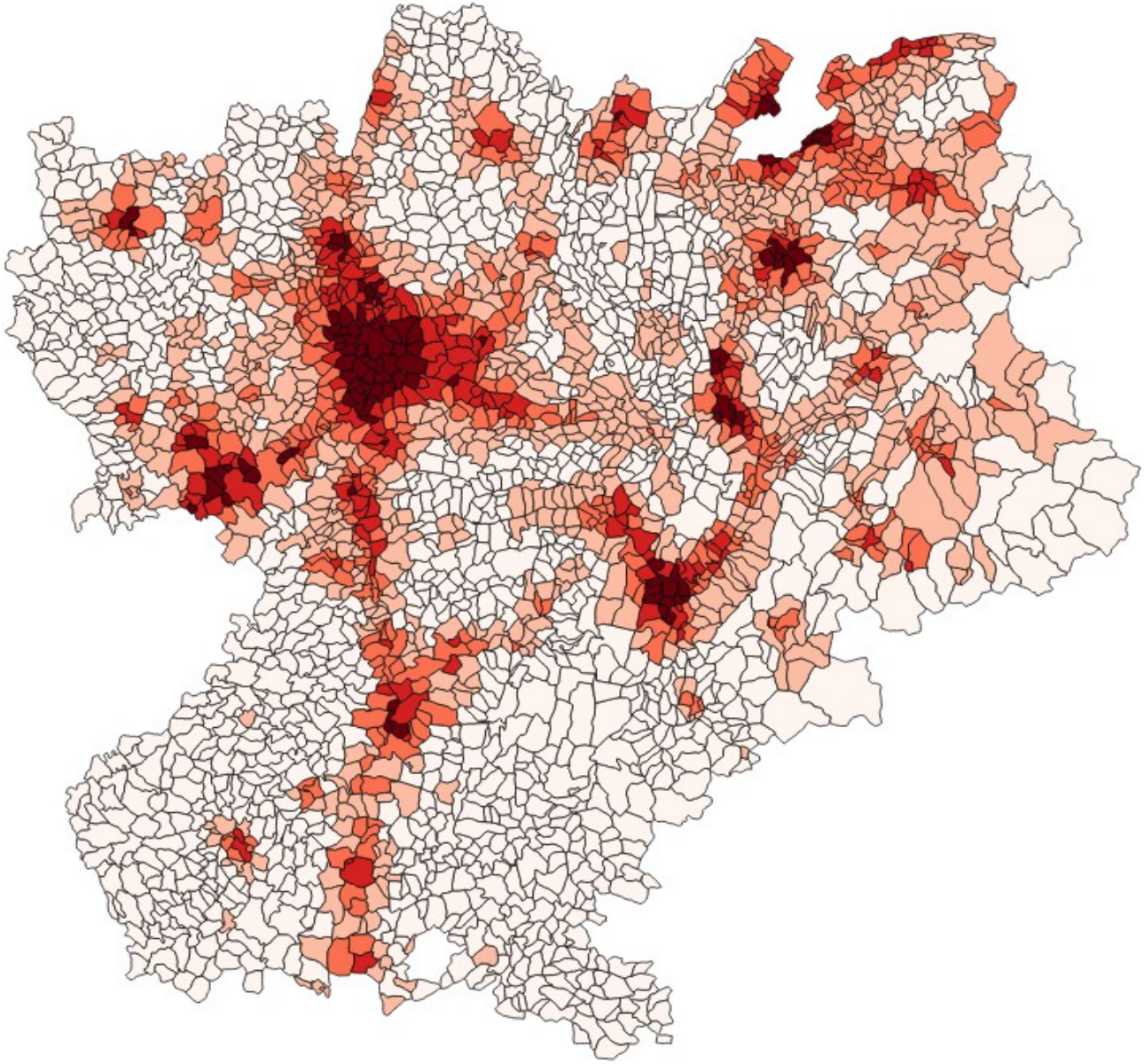


Open Data from : http://geodata.grid.unep.ch - 2003 Raster

Raster (light pollution) / Vector (area) Intersection

```
WITH ln AS
(
   SELECT id, avg(px) AS light

   FROM
   (
      SELECT  id,  ST_Value(rast, ST_SetSrid((ST_Dumppoints(pts)).geom, 2154)) AS px
      FROM (
                  SELECT id, geom AS pts  FROM own.commune
               ) AS t , r
      WHERE ST_Intersects(rast, pts)
    ) AS tt

   GROUP BY id
)

UPDATE own.commune c SET light = ln.light_pollution FROM ln WHERE c.id = ln.id
```

Light pollution by area



Road density by area

```
ALTER TABLE own.commune ADD COLUMN road_density_2016 numeric;


WITH rd AS (

	SELECT c.id,
	   (SUM(ST_Length(  ST_Intersection(c.geom, r.geom)))   /    ST_Area(c.geom)) AS road_density
	FROM own.commune c, osm.roads_2016 r
	WHERE ST_Intersects(c.geom, r.geom)
	GROUP BY c.id

)

UPDATE own.commune c SET road_density_2016 = rd.road_density FROM rd WHERE c.id = rd.id
```

Table 9-50. Aggregate Functions for Statistics

Function	Argument Type	Return Type	Description
corr(*Y*, *X*)	double precision	double precision	correlation coefficient
covar_pop(*Y*, *X*)	double precision	double precision	population covariance
covar_samp(*Y*, *X*)	double precision	double precision	sample covariance
regr_avgx(*Y*, *X*)	double precision	double precision	average of the independent variable (sum(*X*)/*N*)
regr_avgy(*Y*, *X*)	double precision	double precision	average of the dependent variable (sum(*Y*)/*N*)
regr_count(*Y*, *X*)	double precision	bigint	number of input rows in which both expressions are nonnull
regr_intercept(*Y*, *X*)	double precision	double precision	y-intercept of the least-squares-fit linear equation determined by the (*X*, *Y*) pairs
regr_r2(*Y*, *X*)	double precision	double precision	square of the correlation coefficient
regr_slope(*Y*, *X*)	double precision	double precision	slope of the least-squares-fit linear equation determined by the (*X*, *Y*) pairs
regr_sxx(*Y*, *X*)	double precision	double precision	sum(*X*^2) - sum(*X*)^2/*N* ("sum of squares" of the independent variable)
regr_sxy(*Y*, *X*)	double precision	double precision	sum(*X***Y*) - sum(*X*) * sum(*Y*)/*N* ("sum of products" of independent times dependent variable)
regr_syy(*Y*, *X*)	double precision	double precision	sum(*Y*^2) - sum(*Y*)^2/*N* ("sum of squares" of the dependent variable)
stddev(*expression*)	smallint, int, bigint, real, double precision, or numeric	double precision for floating-point arguments, otherwise numeric	historical alias for stddev_samp
stddev_pop(*expression*)	smallint, int, bigint, real, double precision, or numeric	double precision for floating-point arguments, otherwise numeric	population standard deviation of the input values
stddev_samp(*expression*)	smallint, int, bigint, real, double precision, or numeric	double precision for floating-point arguments, otherwise numeric	sample standard deviation of the input values
variance(*expression*)	smallint, int, bigint, real, double precision, or numeric	double precision for floating-point arguments, otherwise numeric	historical alias for var_samp
var_pop(*expression*)	smallint, int, bigint, real, double precision, or numeric	double precision for floating-point arguments, otherwise numeric	population variance of the input values (square of the population standard deviation)
var_samp(*expression*)	smallint, int, bigint, real, double precision, or numeric	double precision for floating-point arguments, otherwise numeric	sample variance of the input values (square of the sample standard deviation)

SELECT corr (pop_density, light)::numeric(4,4) FROM own.commune;

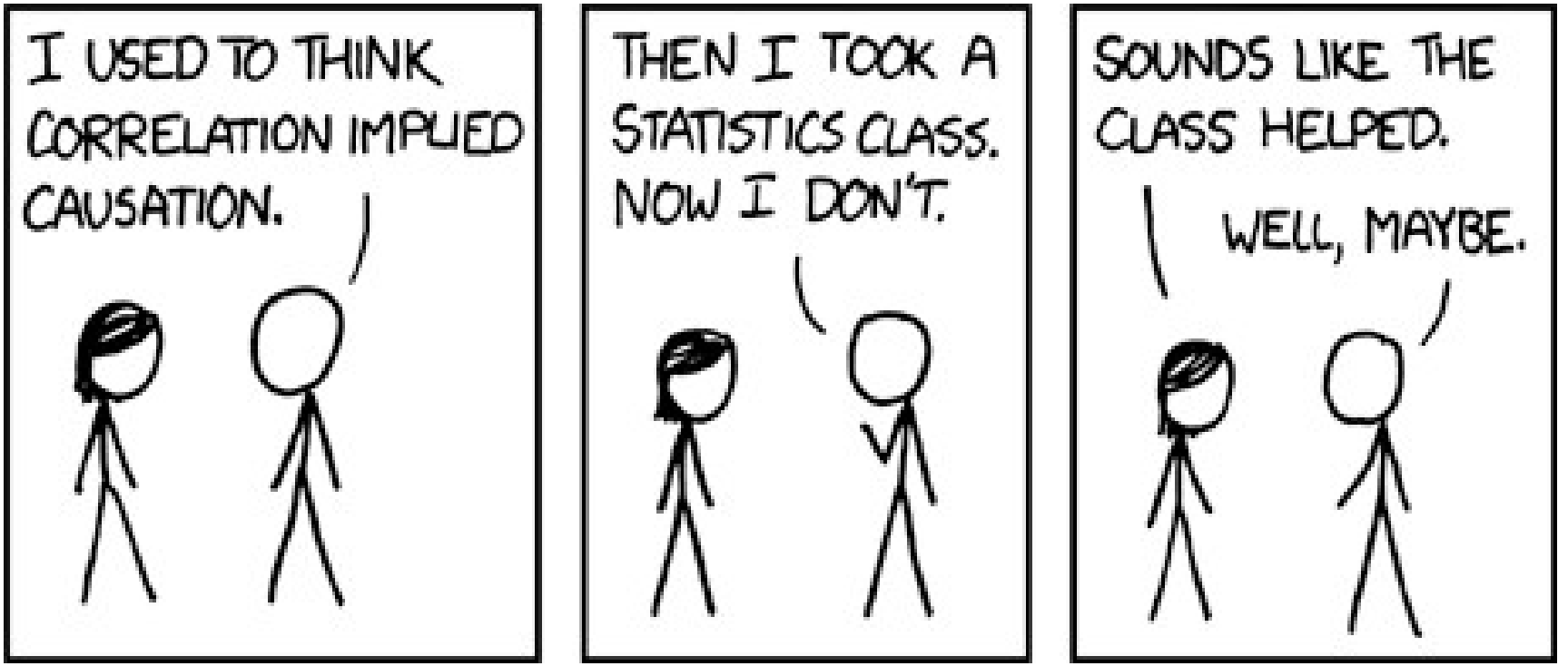
0.6533

- - OSM 08/2014
SELECT corr (road_density, light)::numeric(4,4) FROM own.commune;

0.7573

- - OSM 08/2016
SELECT corr (road_density, light)::numeric(4,4) FROM own.commune;

0.7782



https://en.wikipedia.org/wiki/Correlation_does_not_imply_causation

" Everything is related to everything else,
but near things are more related than distant things. "

W. Tobler

Moran I - Spatial Autocorrelation Coefficient

1 → Strong Spatial Correlation

0 → Random

-1 → Perfectly dispersed

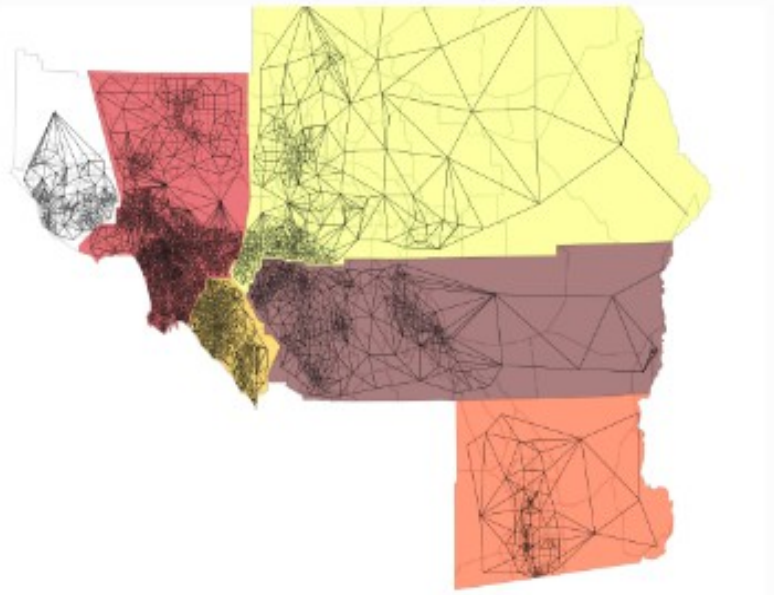
$$I = \frac{N}{\sum_i \sum_j w_{ij}} \frac{\sum_i \sum_j w_{ij}(X_i - \bar{X})(X_j - \bar{X})}{\sum_i (X_i - \bar{X})^2}$$

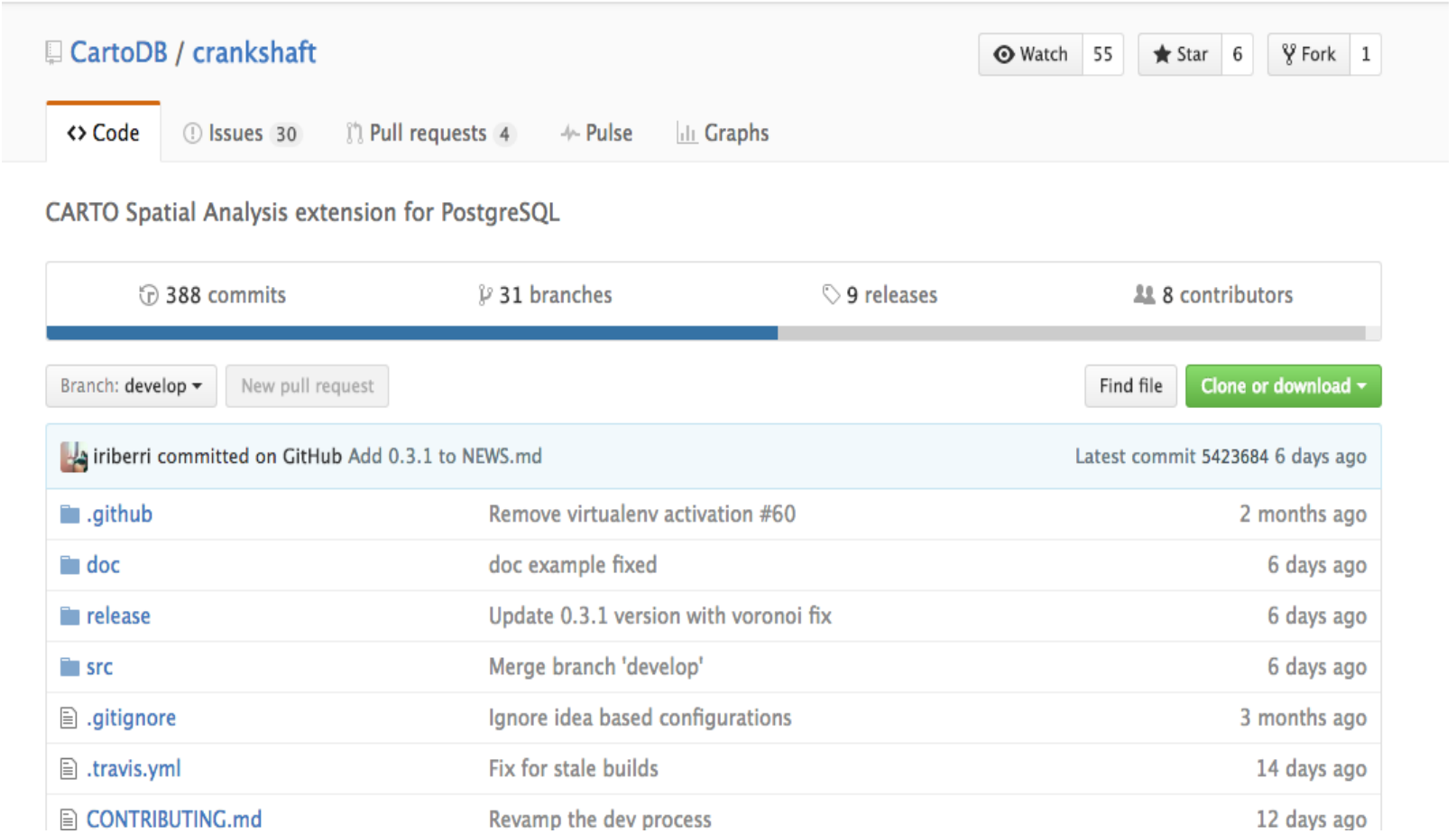
Humm, do we really need R ?

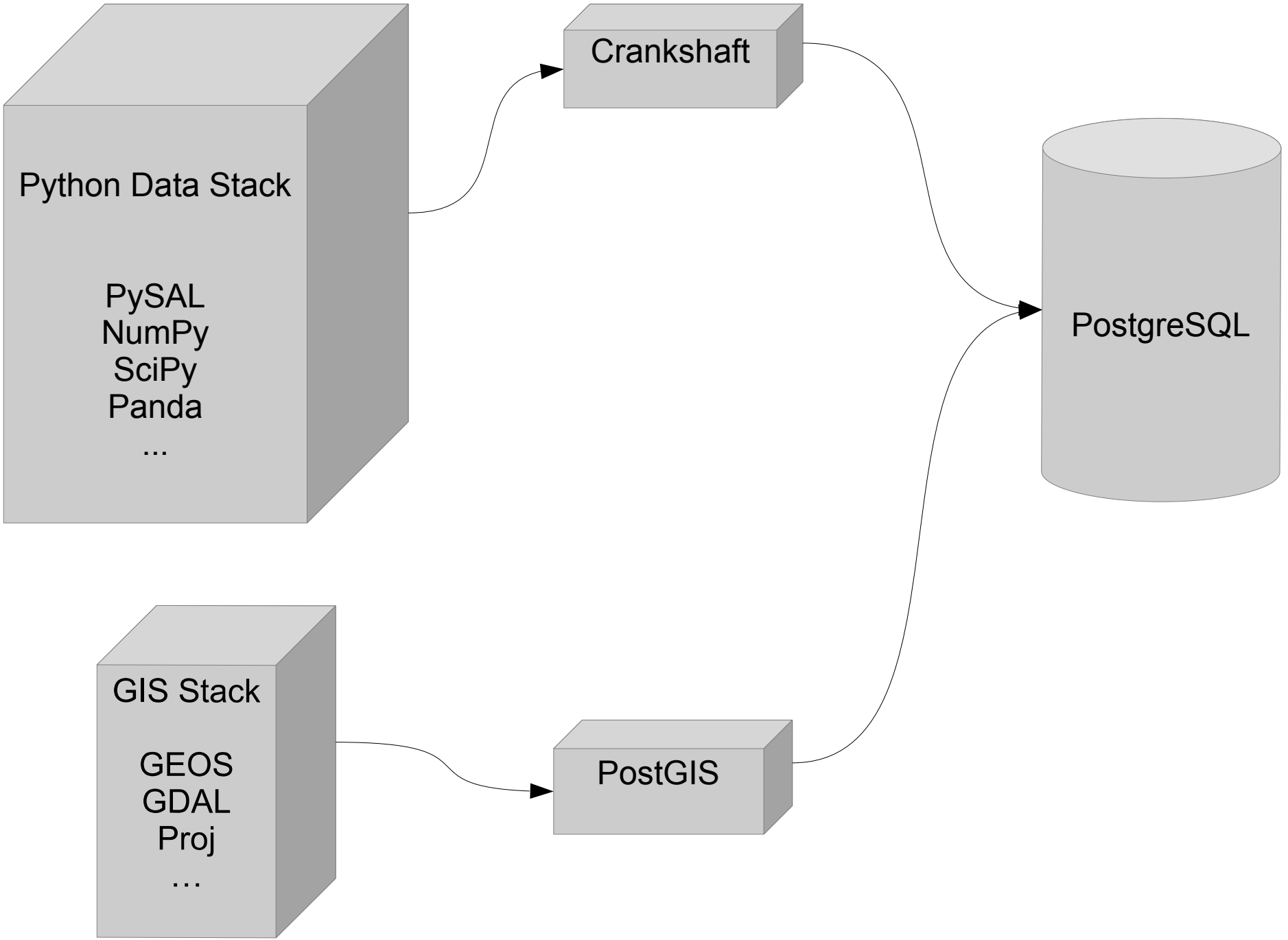
http://pysal.github.io/grid.html

PySAL: Python Spatial Analysis Library

This page collects links to examples using pysal. Click on each figure to see access the full example with code included.



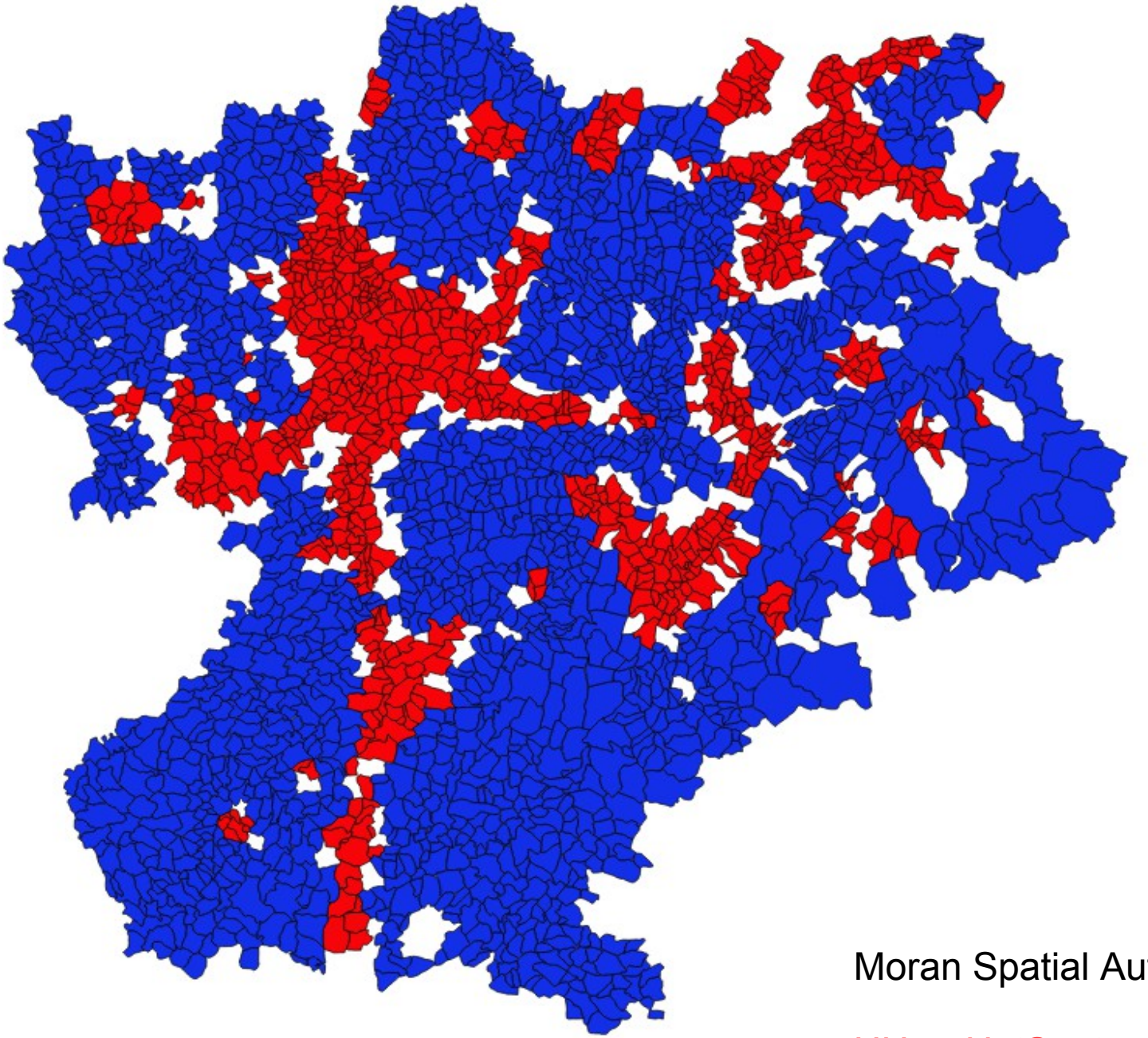

CartoDB / crankshaft
Watch 55
Star 6
Fork 1
Code
Issues 30
Pull requests 4
Pulse
Graphs
CARTO Spatial Analysis extension for PostgreSQL
388 commits
31 branches
9 releases
8 contributors
Branch: develop
New pull request
Find file
Clone or download
iriberri committed on GitHub Add 0.3.1 to NEWS.md
Latest commit 5423684 6 days ago
.github Remove virtualenv activation #60 2 months ago
doc doc example fixed 6 days ago
release Update 0.3.1 version with voronoi fix 6 days ago
src Merge branch 'develop' 6 days ago
.gitignore Ignore idea based configurations 3 months ago
.travis.yml Fix for stale builds 14 days ago
CONTRIBUTING.md Revamp the dev process 12 days ago


Python Data Stack
PySAL
NumPy
SciPy
Panda
...
Crankshaft
PostgreSQL
GIS Stack
GEOS
GDAL
Proj
…
PostGIS

```
SELECT moran::numeric(10, 4)

FROM  cdb_crankshaft.cdb_areasofinterestGlobal(

        'SELECT * FROM own.commune',  - - data table
        'light',    - - column name to check
        'knn',      - - weight : queen or knn
        5,          - - k value (for knn)

        99, 'geom', 'id'
    )
```

queen	0.8235
knn5	0.8201
knn20	0.6687
knn50	0.5220

```
WITH m AS (

  SELECT aoi.*, c.id, c.nom_com , c.geom

  FROM cdb_crankshaft.cdb_areasofinterestlocal(
                    'SELECT * FROM own.commune',
                    'light',
                    'knn',
                    5,
                    99,
                    'geom',
                    'id') As aoi
  JOIN own.commune As c
  ON c.id = aoi.rowid

)

SELECT quads, geom, ow_number() OVER() AS id
FROM u
WHERE quads = 'HH' OR quads = 'LL'
```



Moran Spatial AutoCorrelation

HH -> HotSpot
LL -> LowSpot

#3 To fully play with

SQL++

(Open) Data

PostGIS
ToolBox

Statistical
skills

PG Extension

Python

#Conclusion

PostgreSQL behaves like an extensible
and integrated Framework

(modern) SQL and Python acting as glue languages

Possible Bridge beetween
GIS and Python DataScience communities

Thanks