Is London a Forest?

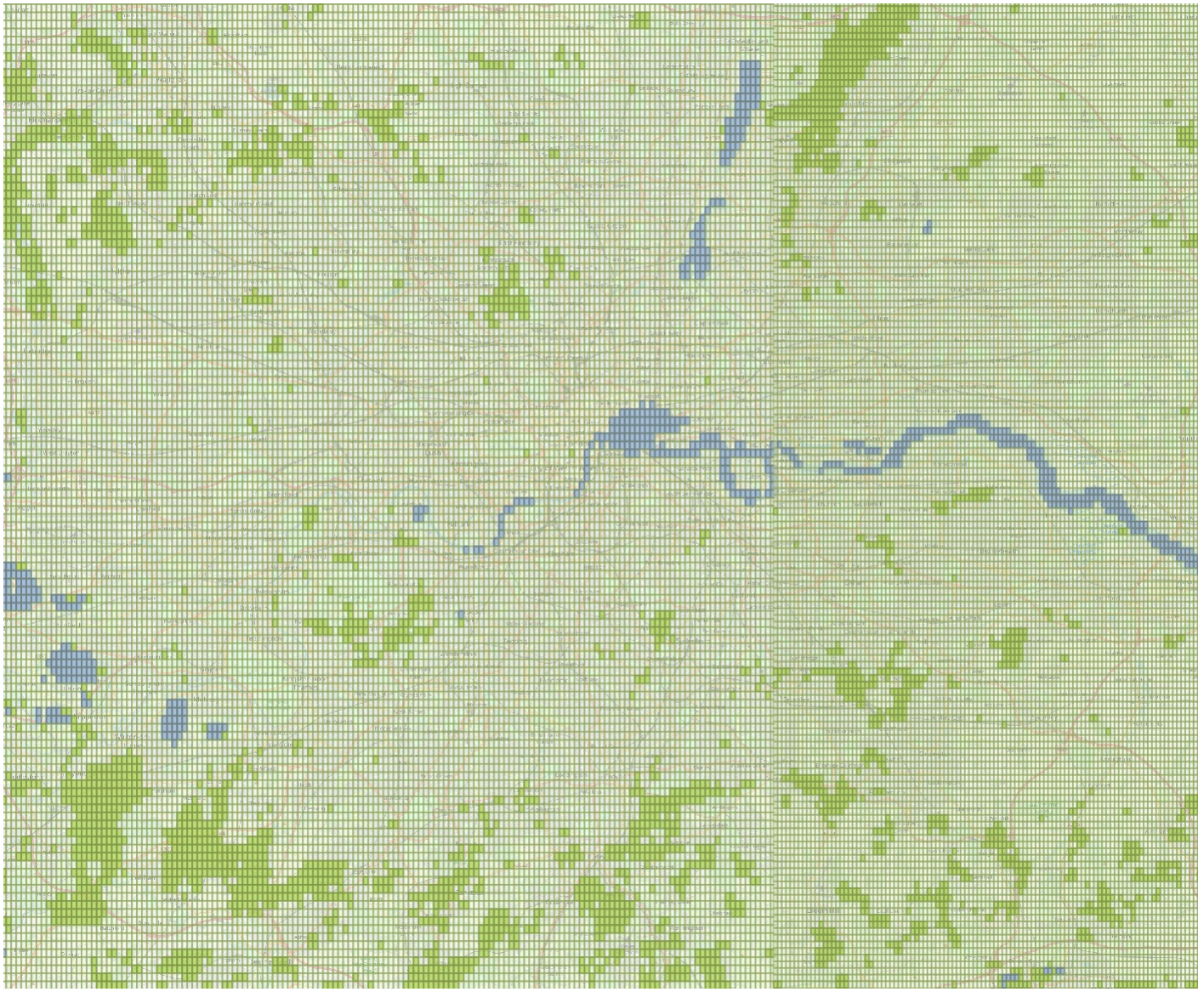
By Ariadni-Karolina Alexiou (carolinegr@gmail.com)

In March 2002, London was classified as an urban forest by the Forestry Commision. According to official sources, 20% (or ~31,000 hectares) of the area of Greater London is covered by tree canopy. To do this classification, ærial images were analyzed using the methodology described here. The methodology is based on multiple humans classifying a random sample of the ærial images and subsequent statistical inference based on that sample. The Greater London Authority states towards the end of the document, that 'it would be useful to compare the findings produced by random point classification of ærial imagery to more automated methods.'

As a data engineer who's especially interested in geospatial data and also really likes forests, I decided to pursue this exact line of research. If you want to follow along fully, check out the project on github.

Datasets

There is a dataset that can be used - NASA's MOD44B V6 dataset which is publicly available for download and fits the requirement of having been annotated using automated methods. This dataset is derived from raw satellite images and stored in HDF format. Each image has information about a part of the Earth. Each pixel in the image represents an area on Earth that is 250m x 250m in size, so it can better be thought of as a polygon on the surface of the Earth rather than a pixel, basically a rectangle. Each rectangle is annotated with metadata - including the percentage of tree cover in the area it covers.



The Greater London Area divided in rectangles with area 250m x 250m - the greener the rectangle, the higher the tree cover percentage

How is the tree cover percentage derived in the NASA dataset?

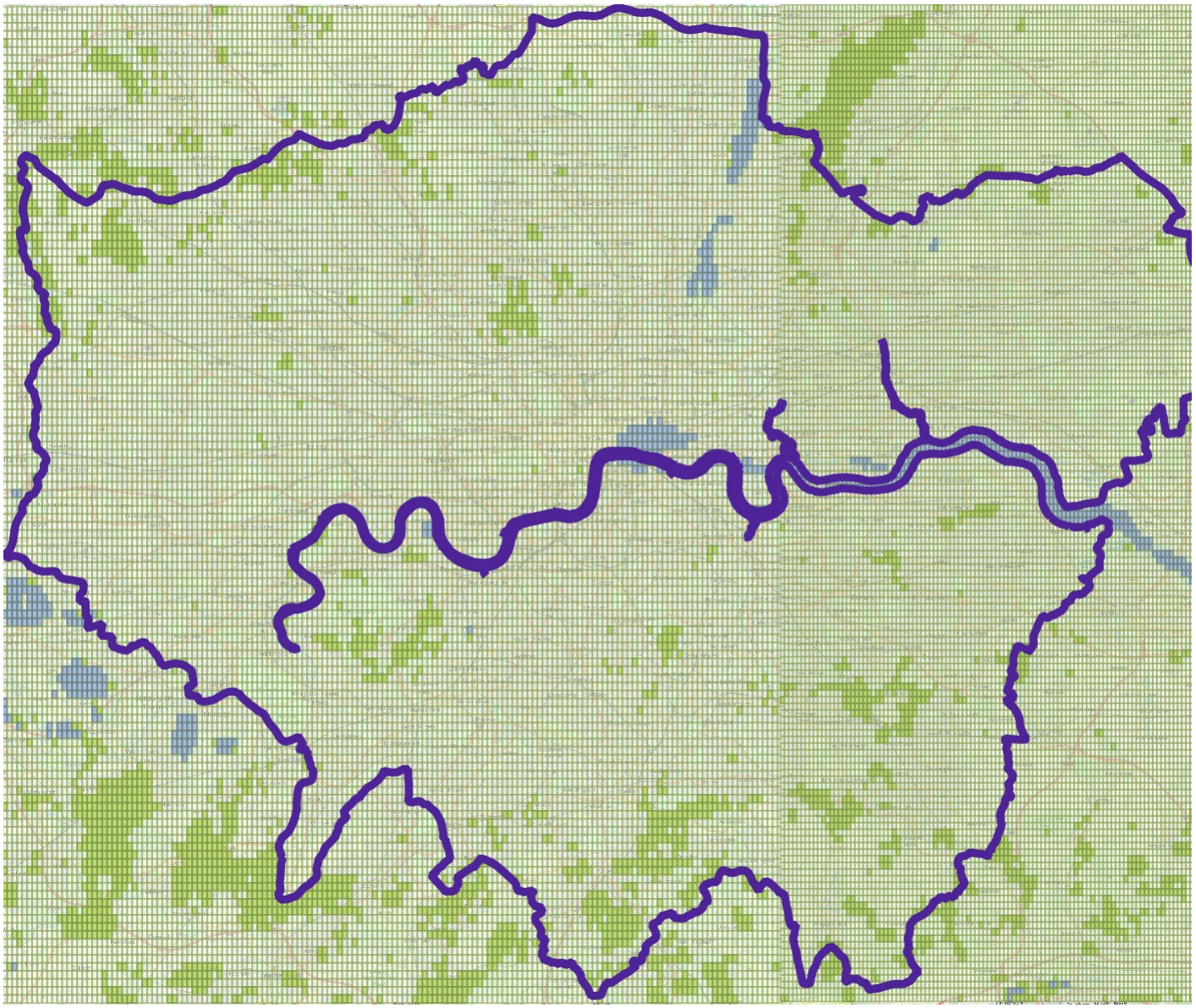
According to the spec, the tree cover is derived by applying a machine learning algorithm on the dataset. Humans classified a set of training rectangles, however, after that, the classification of the rest of the rectangles was done by an algorithmic model built on these training data. This method is quite different than how the classification was done in the original study by the Greater London Authority, which used humans exclusively.

How can we use this dataset?

This dataset gives the tree cover percentage of all surfaces on Earth. We can intersect this dataset with the geographic boundaries of Greater London, and add the areas of all the 250m x 250m rectangles, multiplied by their associated tree cover percentage. The resulting sum will be the forest area of Greater London.

The geographic boundary of Greater London is also publicly available and can be

downloaded here.



The two datasets combined

Processing

We need to obtain the two datasets (NASA satellite images and the geographic boundary of Greater London) and transform them into a common format so that we can do the intersection. People who want to be creative with multiple data sources, for example, data journalists and data scientists, are faced with this challenge all the time. It's very rare that data from diverse sources is directly comparable. Our goal is to turn both datasets into longitudes and latitudes. And then we'll need the right tool to store the data and do the intersection - for which I've decided to use the Postgres database with the Postgis extension.

Prerequisites

To follow the tutorial, you need to check out the code on github, and you also need to have docker and docker-compose installed. Please refer to this tutorial for

installing Docker and this one for docker-compose.

I have created a docker-compose setup that spins up two containers, one with a Postgres database called `londonwald` with the Postgis extensions installed, and one with all the necessary python dependencies installed, which can also access the database container. All you need to do is `docker-compose up -d` in the project's root level directory, and you'll be set to run all the code that follows.

Tools we're going to use

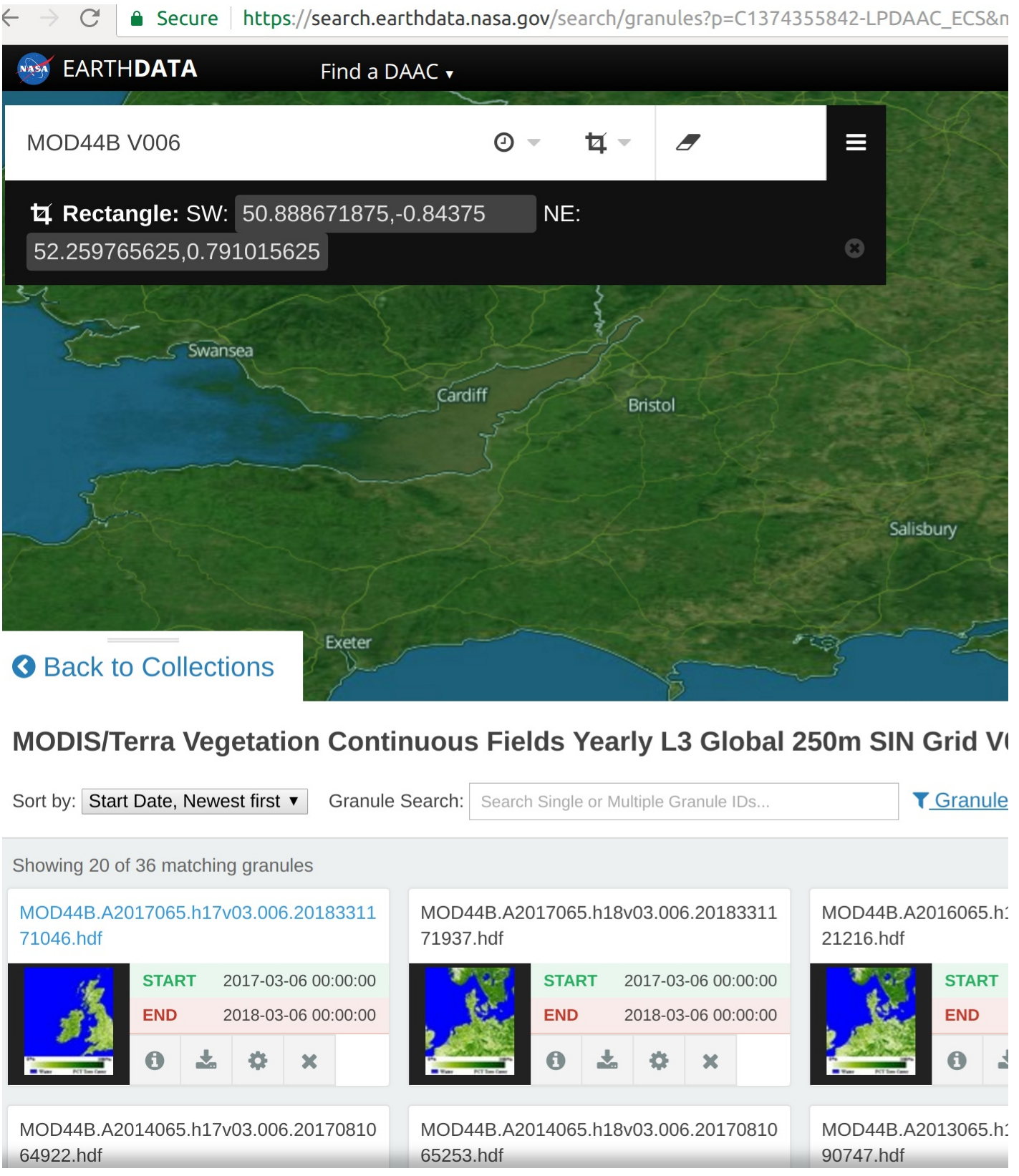
We're going to use Postgres to store the spatial objects we create. The boundary of Greater London is a spatial object, and all the rectangles in the hdf files are also spatial objects. Since we're going to look at a lot of them (around 200,000 rectangles for the Greater London area for the years we are looking at) it makes sense to use a database, rather than try to have everything on memory. In order to make Postgres be able to store spatial objects, in addition to the objects databases usually store (strings, integers and the like), we are going to use the Postgis extension. Postgis also extends the database functionality (joins, grouppings etc) with functions that can deal specifically with spatial objects (spatial joins, intersection, area calculation, dealing with geometries that use different projections) that are both straightforward to use and highly optimized behind the scenes.

We're also going to use the python libraries of GDAL, the Geospatial Data Abstraction Library, which is the work of many years of research, and contains functionality to process all kinds of geospatial datasets, including hdf files.

All these tools are installed automatically when using the docker-compose setup, so you don't need to install them manually - doing `docker-compose up -d` is sufficient.

Loading the NASA dataset to Postgres

The NASA website offers satellite image files (in hdf format) for each year. London, as we can see if we do a bounding box query in the website, spans across two files. To download them from the website, one needs to register - for the purposes of this tutorial, I have downloaded the files for years 2013-2018 in the github repo of the project. To run any of the commands, please set up the containers with `docker-compose up -d`. The first time may take a while to download all the dependencies.



Selecting the hdf files from the NASA website

Now, each hdf file contains a grid with all the 250m x 250m rectangles, and each of these rectangles has values for tree cover percentage.

We have the following three tasks to do:

- The hdf file dœsn't contain just the tree cover percentage, but also other

information, such as cloud cover, quality metrics which we want to skip for the purposes of this tutorial. We want to select just the tree cover information.

This is the command to see all the datasets contained in the hdf file:

```
docker-compose exec python3-container gdalinfo data/
MOD44B.A2016065.h17v03.006.2017081141747.hdf |grep
SUBDATASET
```

The output shows that there's seven datasets stored inside the hdf file, of which the tree cover is the first.

- The hdf file does not use longitudes and latitudes directly - the data is stored in what is called the Sinusoidal projection, as we can learn from the spec. We therefore need to reproject the file to end up with longitudes and latitudes. This will at the same time, reproject the rectangle dimensions from meters to degrees.
- Each point in the grid is a point in a 2D coordinate system from (0,0) to (maxX, max,Y). However this point does actually represent a rectangle whose top left corner is the point in the grid and its width and length are the step sizes in each direction - we need to create a rectangle from this information. Even though the rectangles are in fact squares, because of how longitudes and latitudes work in different places of the Earth, the step sizes in degrees will be different in each direction.

Running the code

We will use the gdal library to do the heavy lifting for the first two tasks, for the third one, we will loop over the grid and create polygons to load onto the database. You can run the code to load the forest rectangles onto the Postgres database by doing:

```
docker-compose exec python3-container python\
 load_hdf_files_to_postgis.py
```

then you can do

```
docker-compose exec python3-container psql
```

and verify that the `forest_boxes` table has been populated, by running the SQL query:

```
SELECT COUNT(*) from forest_boxes;
```

What the code does

By running the code that loads the hdf files onto the Postgres database, we execute several steps. The most important ones are outlined here:

First, we create the database table where the rectangles with tree cover information will be stored:

```
CREATE TABLE IF NOT EXISTS forest_boxes
```

```
(
box geometry,
tree_cover double precision,
year integer
);
```

The geometry type is a data type that is supported by Postgres when Postgis has been installed.

After that, we select the tree cover dataset and reproject it to use longitudes and latitudes:

```
TREE_COVER_DATASET_IDX = 0 # tree cover is the first dataset
hdf_dataset = gdal.Open(hdf_file)
tree_cover_dataset_name = hdf_dataset.GetSubDatasets()\
[TREE_COVER_DATASET_IDX][0]
tree_cover_dataset = gdal.Open(tree_cover_dataset_name,\
 gdal.GA_ReadOnly)
tree_cover_dataset = reproject_to_lonlat(tree_cover_dataset)
```

Then, we extract the metadata from the dataset, to know the width and height of the rectangles, how many there are, and what longitude and latitude the top left rectangle corresponds to:

```
band = tree_cover_dataset.GetRasterBand(1)
band_type = gdal.GetDataTypeName(band.DataType)
geotransform = tree_cover_dataset.GetGeoTransform()
topleftX = geotransform[0]
topleftY = geotransform[3]
stepX = geotransform[1]
stepY = geotransform[5]
X = topleftX
Y = topleftY
```

Then, we connect to the database and prepare the statement to insert each rectangle geometry:

```
conn, cur = get_pg_connection_and_cursor()
insert_sql_statement_template = 'INSERT INTO forest_boxes\
 (box, year, tree_cover) VALUES %s'
```

Finally, we loop through the grid of rectangles, create the descriptions of them in terms of their four corners (longitude, latitude pairs) using the well known text (WKT) format and insert them into the database in batches:

```
from psycopg2.extras import execute_values
for y in range(band.YSize):
scanline = band.ReadRaster(0, y, band.XSize, 1,\
```

```
 band.XSize, 1, band.DataType)
tree_cover_values = struct.unpack(BAND_TYPES_MAP[band_type] *\
 band.XSize, scanline)
tuples = []

for tree_cover_value in tree_cover_values:
if intersects_with_greater_london_area_bounding_box(X, Y,\
 stepX, stepY):
ewkt = get_well_known_text_for_box_geometry(X, Y, stepX,\
 stepY)
tuples.append((ewkt, year, tree_cover_value))
X += stepX

X = topleftX
Y += stepY
if len(tuples) > 0:
# execute_values can insert multiple rows at once,
# faster than doing it one by one
execute_values(cur, insert_sql_statement_template, tuples)

conn.commit()
cur.close()
conn.close()
```

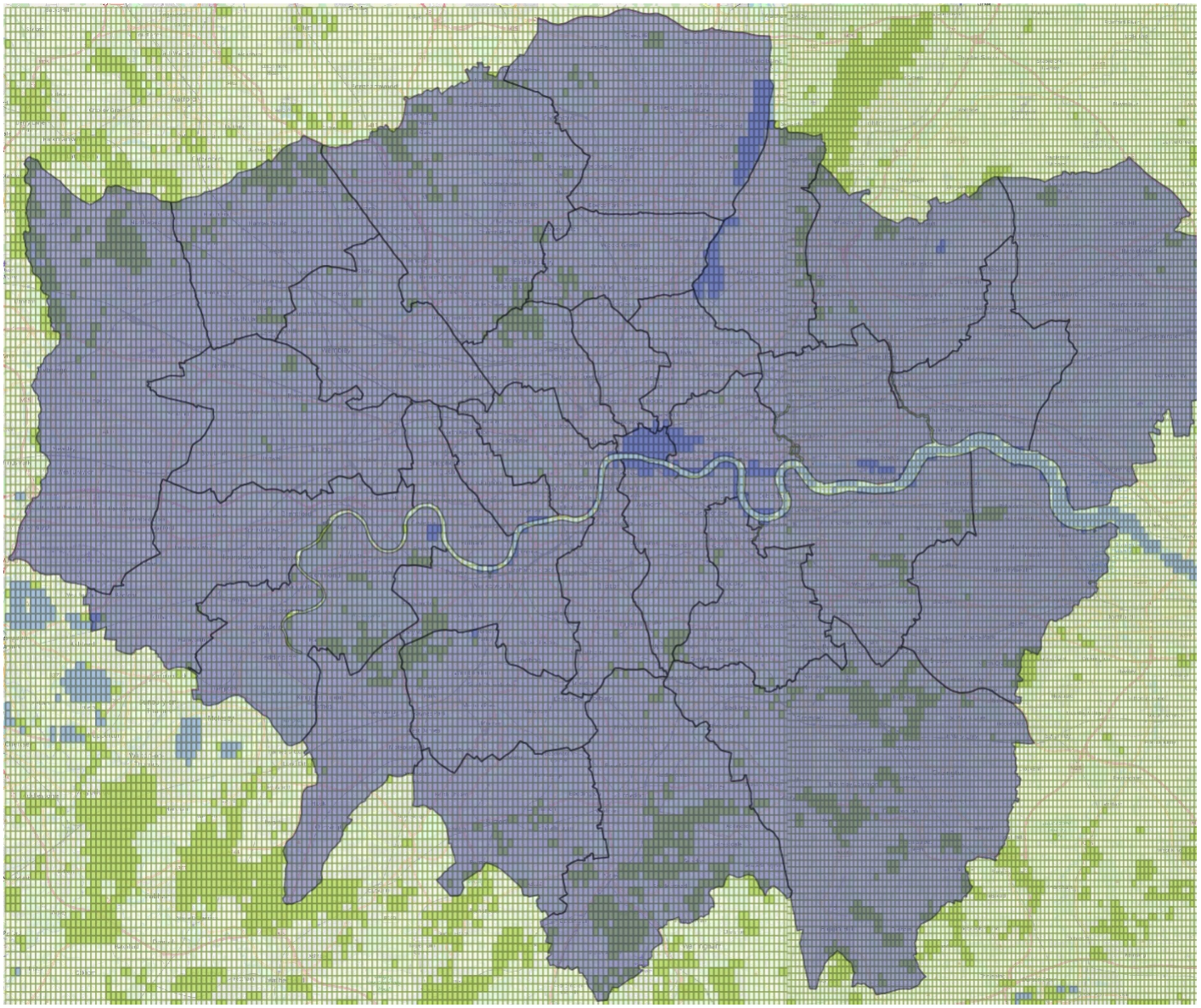
Loading the Greater London Area boundary onto Postgres

I've downloaded the London shapefiles from here and used shp2pgsql, a handy tool that can turn shapefiles into a postgres table. To simplify the tutorial, I have already done this conversion, and the resulting sql file from processing `statistical-gis-boundaries-london/ESRI/London_Borough_Excluding_MHW.shp` can be found on github. Please keep in mind that the polygons of the boroughs are not yet converted into longitudes and latitudes in this file, instead, the projection of the British National Grid(WKID: 27700) is being used.

We can then load the table onto Postgres:

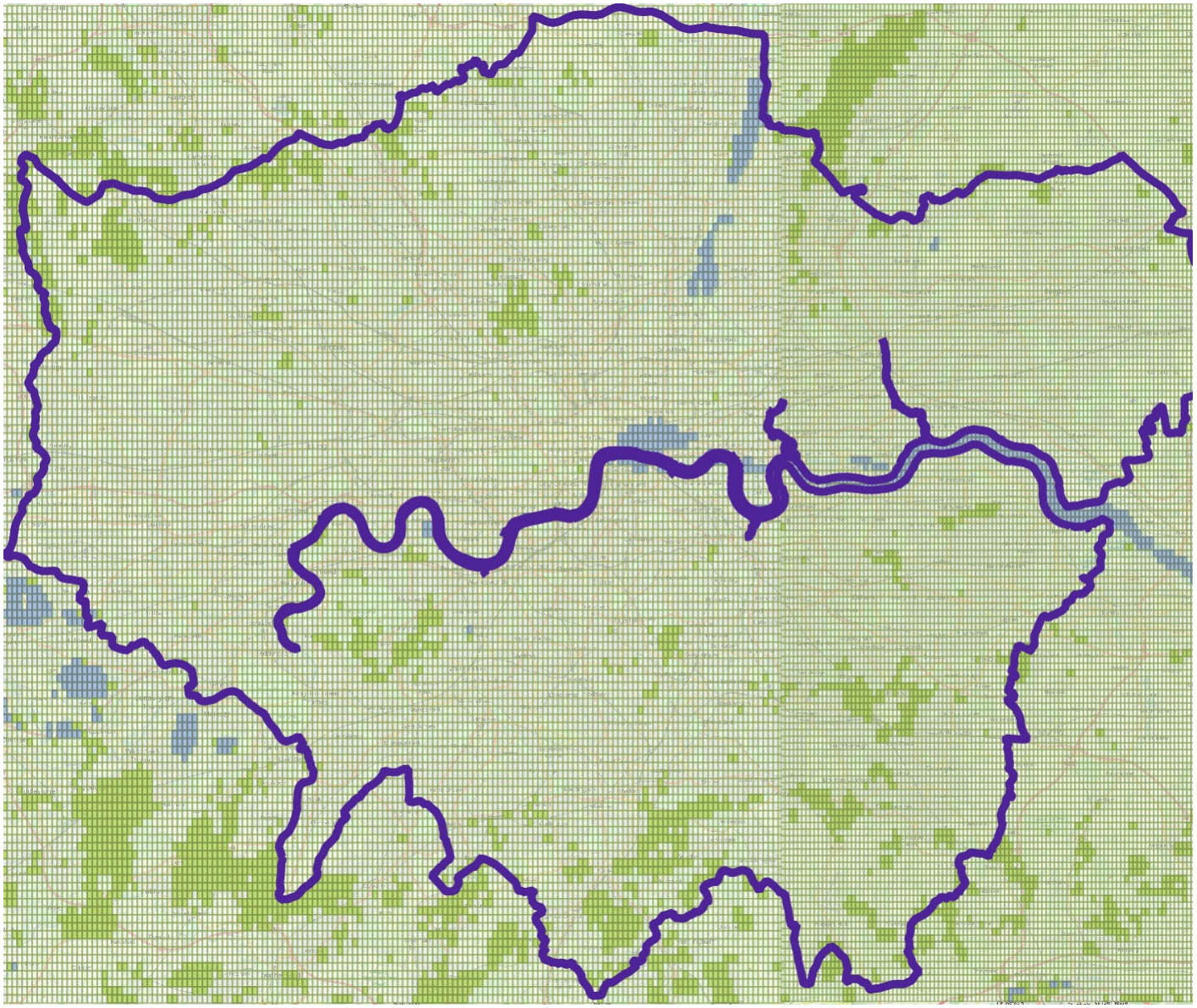
```
docker-compose exec python3-container psql -f\
 data/londonborough.sql
```

Now we have all the boroughs of the Greater London Area loaded, which is a bit more detail than we actually need.



What we actually want is the entire area as a polygon. Postgis offers two functions that can support us: ST_Union, which can stitch together geometries into one by dissolving their boundaries, and ST_Transform, which can reproject a geometry to a different projection ID (in our case, 4326, which corresponds to longitudes and latitudes). With a simple query, we can create the Greater London area as a geometry in a new table named `greaterlondon` (that has just one row).

```
docker-compose exec python3-container psql
create table greaterlondon as select st_transform(st_union(geom),\
 4326) as geom from londonborough;
```



Boundaries dissolved!

Answering the question

Now that the rectangles with the tree cover information and the boundary area are in the same format and in the same database, the question can be answered with one SQL query.

This is how we get the tree cover area in hectares for the year 2013, with a little help from the custom Postgis functions ST_Intersection, ST_Intersects and ST_Area.

We multiply the tree cover of each rectangle that intersects the Greater London Area, with the area of said intersection. When geometries are cast as ::geographies, the ST_Area function conveniently returns the result in square meters.

First, connect to the database by doing `docker-compose exec python3-container psql` and run this query:

```
SELECT
```

```
SUM((tree_cover/100) *
ST_Area(ST_Intersection(greaterlondon.geom,\
 forest_boxes.box)::geography))
/10000 as tree_area_hectares

FROM greaterlondon JOIN forest_boxes on ST_Intersects(geom, box)

WHERE year=2013 and tree_cover<=100;
```

The result is 30,900 hectares, very close to the 31,000 reported by the original study. The original study was also based on imagery captured in a different time of the year in 2013 than when the NASA satellite images were captured on the same year, that is, with possible differences in how green the leaves and the grass looks. That we are so close to the exact number reported is a great result.

Closing thoughts

We answered the question of the percentage of tree cover in London using a different starting dataset and different methodology, and, happily, we reached a result which is very close to the result of the original study - it does confirm that 20% of the area of Greater London is covered with trees. Along the way, we learned more about the hdf data format, how to do transformations on geometries, and how to use the Postgis functions to get quick answers about the relationships between them.

We have created a dataset which can have more applications, in addition to helping us classify London as an urban forest. Looking at the distribution of tree cover, for example, per borough, the relevant authorities can decide where to plant more trees. Looking at the evolution of tree cover over time, trends can be discovered. As a rule, what can be measured can be improved.

There's more questions waiting to be asked. The code and datasets (including more years) are available to explore and tinker with on github.