SQL

-- Database: floodpronearea

-- DROP DATABASE IF EXISTS floodpronearea;

CREATE DATABASE floodpronearea

WITH

OWNER = postgres

ENCODING = 'UTF8'

LC\_COLLATE = 'English\_United Kingdom.1252'

LC\_CTYPE = 'English\_United Kingdom.1252'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1

IS\_TEMPLATE = False;

-- Table: public.London\_Borough\_Excluding\_MHW

-- DROP TABLE IF EXISTS public."London\_Borough\_Excluding\_MHW";

CREATE TABLE IF NOT EXISTS public."London\_Borough\_Excluding\_MHW"

(

id integer NOT NULL DEFAULT nextval('"London\_Borough\_Excluding\_MHW\_id\_seq"'::regclass),

geom geometry(MultiPolygon),

borough\_name character varying(22) COLLATE pg\_catalog."default",

gss\_code character varying(9) COLLATE pg\_catalog."default",

hectares double precision,

nonld\_area double precision,

ons\_inner character varying(1) COLLATE pg\_catalog."default",

sub\_2009 character varying(7) COLLATE pg\_catalog."default",

sub\_2006 character varying(10) COLLATE pg\_catalog."default",

CONSTRAINT "London\_Borough\_Excluding\_MHW\_pkey" PRIMARY KEY (id)

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."London\_Borough\_Excluding\_MHW"

OWNER to postgres;

-- Creating a Spatial Database

--

CREATE EXTENSION postgis;

SELECT postgis\_full\_version();

**Opening the shapefile using the postgis**

select \*

from public."London\_Borough\_Excluding\_MHW"

**Opening the tabular file using the postgis**

SELECT \* FROM public."london borough data"

ORDER BY id ASC

**Using inner join to connect the tables**

select \*

from public."london borough data"

inner join public."London\_Borough\_Excluding\_MHW"

on public."london borough data".id=public."London\_Borough\_Excluding\_MHW".id

**Selecting the columns from both the tables**

select "London\_Borough\_Excluding\_MHW".name,"London\_Borough\_Excluding\_MHW".geom,"london borough data".area\_name,"london borough data".population

from public."london borough data"

inner join public."London\_Borough\_Excluding\_MHW"

on public."london borough data".id=public."London\_Borough\_Excluding\_MHW".id

Distance of river thames

SELECT ST\_Distance(

'SRID=4326;POINT(-0.616075 51.585574)'::geometry, -- river thames

'SRID=4326;POINT(-0.100594 51.376495)'::geometry -- croydon

);

SELECT ST\_Area(geom)

FROM public."London\_Borough\_Excluding\_MHW"

CREATE TABLE buffer\_area AS

SELECT

ST\_Intersection(

ST\_Buffer(ps.geom, 50),

ST\_Buffer(cg.geom, 50))

AS geom

FROM

public."London\_Borough\_Excluding\_MHW" ps,

public."London\_Borough\_Excluding\_MHW" cg

WHERE ps.borough\_name = 'croydon'

AND cg.borough\_name = 'brent';

Python

!pip install osmnx scipi xarray rioxarray matplotlib datashader -q

pip install xarray-spatial -q

from pathlib import Path

from IPython.core.display import Video

import numpy as np

import pandas as pd

import geopandas as gpd  # Vector data handling

import osmnx as ox       # Downloading data from OSM

from shapely.geometry import box

from scipy.spatial import cKDTree as KDTree # For Inverse Distance Weight calculation

import xarray as xr

import xrspatial    # Hillshading

import rioxarray   # Working with geospatial data in xarray

import matplotlib.pyplot as plt

from datashader.transfer\_functions import shade, stack

dem = rioxarray.open\_rasterio('/content/data\_3.tif')

!pip install geojson

Collecting geojson

Downloading geojson-3.0.1-py3-none-any.whl (15 kB)

Installing collected packages: geojson

Successfully installed geojson-3.0.1

geom = '''{"type": "Polygon",

                "coordinates":[[[-0.7667820837,51.2231086713],[0.4118149096,51.2231086713],[0.4118149096,51.7310469889],[-0.7667820837,51.7310469889],[-0.7667820837,51.2231086713]]]}'''

raster\_file = rioxarray.open\_rasterio("data\_3.tif", masked=True).squeeze()

raster\_file = raster\_file.coarsen(x=3, boundary='trim').mean().coarsen(y=3, boundary='trim').mean()

raster\_file.squeeze().plot.imshow()

river = ox.geocode\_to\_gdf('Thames River', which\_result=1)

river = river.to\_crs(raster\_file.rio.crs)

river.plot()

raster\_file.rio.bounds()

bounds = raster\_file.rio.bounds()

xmin, ymin, xmax, ymax = bounds

river = river.clip(bounds)

river\_geom = river.geometry.iloc[0]

river\_geom

raster\_file = raster\_file.sel(y=slice(ymax, ymin), x=slice(xmin, xmax))

fig, ax = plt.subplots()

raster\_file.squeeze().plot.imshow(ax=ax)

river.plot(ax=ax, color='red')

import shapely

def split\_coords(geom):

    x = []

    y = []

    for i in shapely.get\_coordinates(geom):

        x.append(i[0])

        y.append(i[1])

    return x, y

xs, ys = split\_coords(river\_geom)

xs, ys = xr.DataArray(xs, dims='z'), xr.DataArray(ys, dims='z')

sampled = dem.interp(x=xs, y=ys, method='nearest').dropna(dim='z')

# Sampled river coordinates

c\_sampled = np.vstack([sampled.coords[c].values for c in ('x', 'y')]).T

# All (x, y) coordinates of the original DEM

c\_x, c\_y = [dem.coords[c].values for c in ('x', 'y')]

c\_interpolate = np.dstack(np.meshgrid(c\_x, c\_y)).reshape(-1, 2)

# Sampled values

values = sampled.values.ravel()

tree = KDTree(c\_sampled)

# IWD interpolation

distances, indices = tree.query(c\_interpolate, k=5)

weights = 1 / distances

weights = weights / weights.sum(axis=1).reshape(-1, 1)

interpolated\_values = (weights \* values[indices]).sum(axis=1)

elevation\_raster = xr.DataArray(

    interpolated\_values.reshape((len(c\_y), len(c\_x))).T, dims=('x', 'y'), coords={'x': c\_x, 'y': c\_y}

)

fig, ax = plt.subplots()

elevation\_raster.transpose().plot.imshow(ax=ax)

river.plot(ax=ax, color='red')

rem = dem - elevation\_raster

colors = ['#f2f7fb', '#81a8cb', '#37123d']

shade(rem.squeeze(), cmap=colors, span=[0, 10], how='linear')

a = shade(xrspatial.hillshade(dem.squeeze(), angle\_altitude=1, azimuth=310), cmap=['black', 'white'], how='linear')

b = shade(rem.squeeze(), cmap=colors, span=[0, 10], how='linear', alpha=200)

stack(a, b)