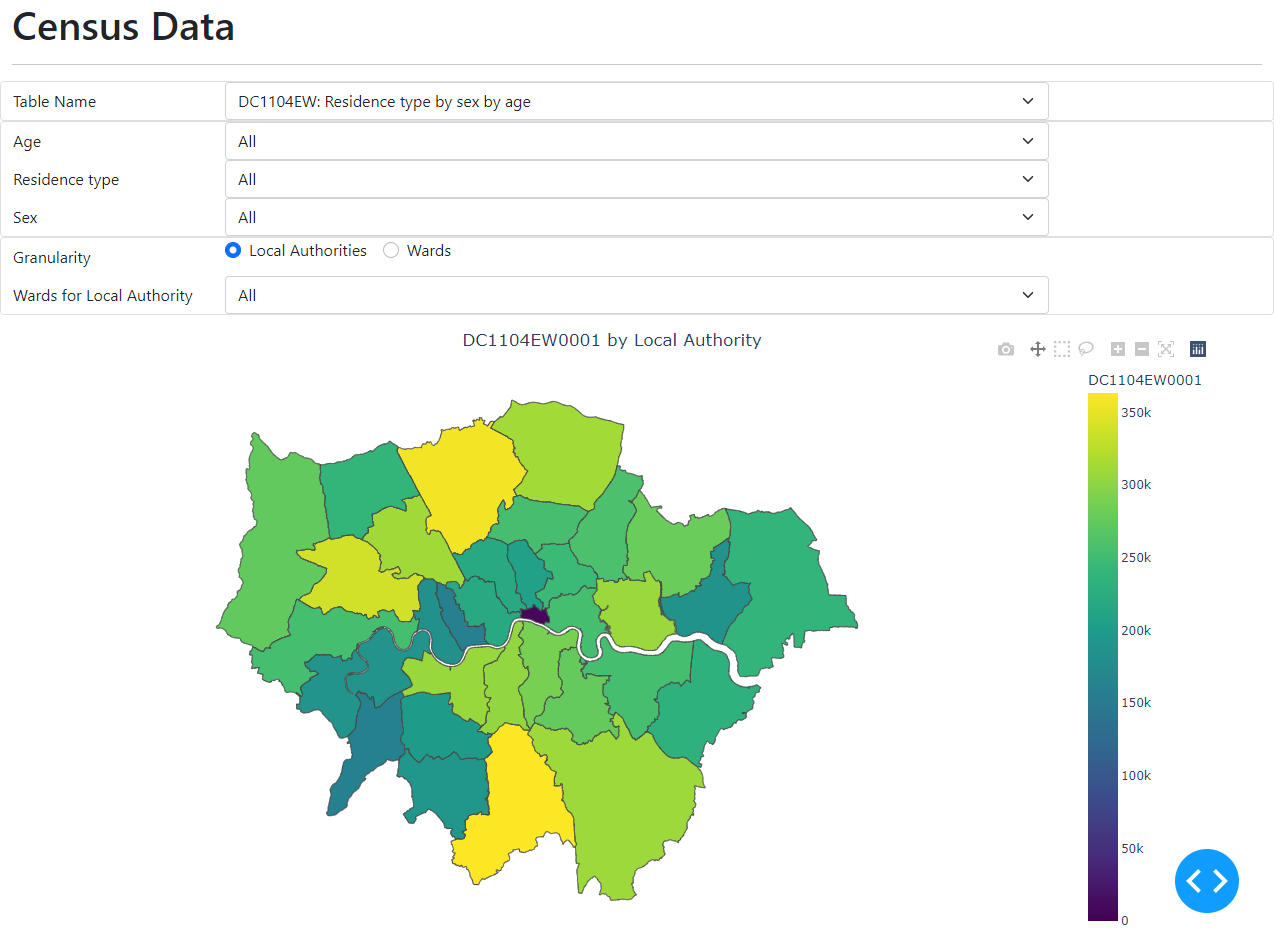
Interactive Mapping in Python with UK Census Data

# Introduction

In this article I will describe the process I followed to create this dashboard displaying maps of London showing UK Census Data:



I am new to GIS and mapping, so this was a voyage of discovery – I will describe what I learned, and some of the issues I encountered along the way, with their resolutions.

I assume familiarity with Python. The most common Python data visualization libraries are compared in this article [The Top 6 Python Data Visualization Libraries: How to choose](https://www.justintodata.com/python-data-visualization-libraries/). Their summary was:

* [Matplotlib](https://matplotlib.org/) - foundation and comprehensive library, highly customizable, not very easy to learn and use
* [seaborn](https://seaborn.pydata.org/) - easy to learn and use, good for exploratory data analysis, relatively customizable, can integrate with Matplotlib if need more customizations
* [Plotly](https://plotly.com/python/) ([Dash](https://dash.plotly.com/)) - easy to use and customizable, create interactive graphs and web applications
* [Bokeh](https://bokeh.org/) - create interactive graphs and web applications
* [folium](https://pypi.org/project/folium/) - good for nice looking map graphs
* [Plotnine](https://pypi.org/project/plotnine/) - grammar of graphics, based on the R ggplot2 library

The steps I followed, which will be described in detail below, were:

1. Download UK Census Data and Census Ward and Local Authority geographic information (as Shapefiles).
2. Plot a simple fixed map using GeoPandas and Matplotlib.
3. Change to use Plotly with GeoJSON data
4. Add interactive functionality using Dash

My code is available in this [Github repository](https://github.com/pohara60/Census-Mapping). When I show code fragments below I will refer to the file in the repository where it appears.

# UK Census Data

## Data

The UK Office for National Statistics is responsible for the Census. The 2021 Census results are still being prepared, the schedule is discussed [here](https://blog.ons.gov.uk/2021/10/12/census-2021-the-count-is-done-the-data-is-in-so-what-happens-next/). The 2011 Census data is described and can be downloaded [here](https://www.ons.gov.uk/census/2011census/2011censusdata). For this exercise, I downloaded Bulk Data which is described as:

Bulk data products enable users, particularly managers of information systems, to load large volumes of tables into their own systems in automated ways.

The data is compiled by various areas:

Datasets available in standard bulk format generally contain data only for output areas, lower and middle layer super output areas, wards, local authorities, unitary authorities, counties, regions, and countries.

I downloaded the [Detailed Characteristics on demography and families for merged wards, local authorities and regions](http://webarchive.nationalarchives.gov.uk/20160110193526/http:/data.statistics.gov.uk/Census/BulkdatadetailedcharacteristicsmergedwardspluslaandregE&Wandinfo3.3.zip). This is a 79MB Zip file, containing data in a structured CSV format. I extracted the files into my project *data* directory. Let’s look at the code to read the data.

### Tables

The different tables are described in *Cell Numbered DC Tables 3.3.xlsx*. The *Index* sheet of this file lists the tables, and I read it as follows. (This code appears in the file census\_read\_data.py in the repository.)

import pandas as pd

CENSUS\_DATA = 'data/BulkdatadetailedcharacteristicsmergedwardspluslaandregE&Wandinfo3.3'

f = None

def read\_index():

# Read Index sheet

    global f

    excel\_file = CENSUS\_DATA+'/Cell Numbered DC Tables 3.3.xlsx'

    f = pd.ExcelFile(excel\_file)

    index = f.parse(sheet\_name='Index')

    return index

The *index* DataFrame looks like this:

Table Number Table Title  
0 DC1104EW DC1104EW: Residence type by sex by age  
1 DC1106EW DC1106EW: Schoolchildren and full-time student...  
2 DC1107EW DC1107EW: Marital and civil partnership status...  
3 DC1108EW DC1108EW: Living arrangements by sex by age  
4 DC1109EW DC1109EW: Household composition by age by sex

There are 25 tables of different statistics, each has its own sheet. Looking at the sheet for the first table *DC1104EW*, we see:



The table has categories for *Residence Type*, *Sex* and *Age*. The values in the table are the index of the column in the data file for the statistic. The number of categories varies for each table, with one or two categories in columns, and one or two in the rows. I could not find a way to automatically cope with these varied column and row headings using pandas. Instead, I read the sheet into a DataFrame and then process the headings in my own code, part of which is shown below.

def read\_table(table\_name):

    # Read Table sheet, transforming it into a DataFrame with columns

    # for each category and the Dataset index

    if f == None:

        exit("Call read\_index() to open table list")

    table = f.parse(sheet\_name=table\_name, header=None)

    # Lots of code to process headings not shown here, see

# census\_read\_data.py. Constructs these variables:

# col\_level\_names - names of the column categories

# col\_level\_values - values of the column categories for each column

# row\_level\_names - names of the row categories

# row\_level\_values - values of the row categories for each row

    # Construct DataFrame from column and row level names and values

    num\_cols = len(col\_level\_values[0])

    num\_rows = len(row\_level\_values[0])

    for l in range(row\_levels):

        # Repeat row values in order

        row\_level\_values[l] = [x for x in row\_level\_values[l]

                               for n in range(num\_cols)]

    for l in range(col\_levels):

        # Repeat col values in turn

        col\_level\_values[l] = col\_level\_values[l] \* num\_rows

    values = []

    for r in range(0, num\_rows):

        row = data\_row\_indexes[r]

        values.extend(table.iloc[row, 1:])

    values = [str(int).zfill(4) for int in values]

    data = [row\_level\_values[l] for l in range(row\_levels)] + \

           [col\_level\_values[l] for l in range(col\_levels)] + \

        [values]

    index = row\_level\_names+col\_level\_names+['Dataset']

    df = pd.DataFrame(

        data=data,

        index=index

    )

    df = df.transpose()

    return df

Calling read\_table for DC1104EW gives this DataFrame:

Age Residence type Sex Dataset  
0 All All All 0001  
1 All All Males 0002  
2 All All Females 0003  
3 All Lives in a household All 0004  
4 All Lives in a household Males 0005  
...

Each row in the DataFrame identifies the Dataset for a combination of the category values. The first row, with the value All for the three categories identifies Dataset 0001, which corresponds to the column DC1104EW0001 in the data file DC1104EWDATA.CSV.

### Data Files

Each table has many CSV files; the file we load for DC1104EW is DC1104EWDATA.CSV. As explained above, this data has columns for the different combinations of the category values. The rows have the counts for different geographical areas. We read the data as follows:

def read\_data(table\_name):

    datafile = CENSUS\_DATA + '/' + table\_name + 'DATA.CSV'

    df = pd.read\_csv(datafile)

    return df

The head of the DataFrame for table DC1104EW is:

GeographyCode DC1104EW0001 DC1104EW0002 DC1104EW0003 ... DC1104EW0198  
0 K04000001 56075912 27573376 28502536 ... 156146  
1 E92000001 53012456 26069148 26943308 ... 146915  
2 W92000004 3063456 1504228 1559228 ... 9231  
3 E12000001 2596886 1269703 1327183 ... 7608  
4 E12000002 7052177 3464685 3587492 ... 20369

## Geography

The census data is summarised by Merged Ward and by Local Authority District. Merged Wards refer to Electoral Wards, where a number of small wards are merged to protect privacy. Merged Wards are assigned to Local Authorities, which are themselves assigned to Regions. The geography data is published on the [Open Geography Portal](https://geoportal.statistics.gov.uk/) in a number of formats:

* [Shapefile](https://en.wikipedia.org/wiki/Shapefile) – A geospatial vector data format for geographic information system (GIS) software. It is developed and regulated by [Esri](https://en.wikipedia.org/wiki/Esri).
* [GeoJSON](https://en.wikipedia.org/wiki/GeoJSON) – An open standard format designed for representing simple geographical features, along with their non-spatial attributes, based on JSON.
* [KML](https://en.wikipedia.org/wiki/Keyhole_Markup_Language) - Keyhole Markup Language is an XML format developed for use with Google Earth.

The Shapefile format is much more compact than GeoJSON, and is supported by [GeoPandas](https://geopandas.org/en/stable/), (see below), so this is what I chose to download. (Plotly requires GeoJSON, which I created from the Shapefiles later.)

The geography data for Wards and Local Authority Districts (LADs) that I used is on the [Open Geography Portal](https://geoportal.statistics.gov.uk/) under the menu options *Boundaries | Census Boundaries | Census Merged Wards* and *Boundaries | Administrative Boundaries | Local Authority Districts*. The files I downloaded were:

1. [Census\_Merged\_Wards\_(December\_2011)\_Boundaries](https://geoportal.statistics.gov.uk/datasets/census-merged-wards-december-2011-full-extent-boundaries-in-england-and-wales/explore?location=52.950000%2C-2.000000%2C7.23) – Download Shapefile format
2. [Local\_Authority\_Districts\_(December\_2011)\_Boundaries\_EW\_BFC](https://geoportal.statistics.gov.uk/datasets/local-authority-districts-december-2011-boundaries-ew-bfc/explore?location=52.950000%2C-2.000000%2C7.23) – Download Shapefile format
3. [Ward\_to\_Census\_Merged\_Ward\_to\_Local\_Authority\_District\_(December\_2011)\_Lookup\_in\_England\_and\_Wales](https://geoportal.statistics.gov.uk/datasets/ward-to-census-merged-ward-to-local-authority-district-december-2011-lookup-in-england-and-wales/explore) – Download CSV format

The Shapefiles I downloaded are high resolution, so large: about 120MB and 40MB respectively. The portal has lower resolution versions that are a tenth of the size if you prefer to use those. (Alternatively, you could use a site like [mapshaper.org](https://mapshaper.org/) to simplify the files.)

These Shapefiles have Coordinate Reference System “OSGB36 / British National Grid”. In order to map the data we need to change it to EPSG 4326 (aka WGS84), we will see this in the code below. (I am afraid that it took me a long frustrating time to find this out!)

I used the lookup CSV file to create a list geography lookup DataFrame, with rows for Merged Wards and Local Authority Districts:

def read\_geography():

    # Get Census Merged Ward and Local Authority Data

    lookupfile = 'data/Ward\_to\_Census\_Merged\_Ward\_to\_Local\_Authority\_District\_(December\_2011)\_Lookup\_in\_England\_and\_Wales.csv'

    cmwd = pd.read\_csv(lookupfile, usecols=[

        'CMWD11CD', 'CMWD11NM', 'LAD11CD', 'LAD11NM'])

    cmwd.drop\_duplicates(inplace=True)

    locationcol = "GeographyCode"

    cmwd[locationcol] = cmwd['CMWD11CD']

    namecol = 'Name'

    cmwd[namecol] = cmwd['CMWD11NM']

    lad = pd.read\_csv(lookupfile, usecols=['LAD11CD', 'LAD11NM'])

    lad = lad.drop\_duplicates()

    lad[locationcol] = lad['LAD11CD']

    lad[namecol] = lad['LAD11NM']

    lad['CMWD11CD'] = ''

    lad['CMWD11NM'] = ''

    geography = pd.concat([cmwd, lad])

    return geography

# GeoPandas and Matplotlib

Having downloaded the data, we are ready to produce our first map! While I downloaded data for the whole of England and Wales, I will restrict the mapping to London for simplicity.

[GeoPandas](https://geopandas.org/en/stable/) is an open source project to make working with geospatial data in python easier. GeoPandas extends the datatypes used by [pandas](http://pandas.pydata.org/) to allow spatial operations on geometric types. Geometric operations are performed by [shapely](https://shapely.readthedocs.io/). Geopandas further depends on [fiona](https://fiona.readthedocs.io/) for file access and [matplotlib](http://matplotlib.org/) for plotting.

I installed GeoPandas on Windows and it failed. Because it depends on packages that are implemented in C/C++, special procedures are required to install it on Windows, these are described in the Appendix. (Apparently the install is straightforward on Linux and Mac.)

First, we I load the Shapefile. (Code is in census\_read\_geopandas.py.)

import geopandas as gpd

def read\_london\_lad\_geopandas():

    # Get Local Authority Boundaries as GeoPandas

    shapefile = 'data/Local\_Authority\_Districts\_(December\_2011)\_Boundaries\_EW\_BFC/Local\_Authority\_Districts\_(December\_2011)\_Boundaries\_EW\_BFC.shp'

    gdf = gpd.read\_file(shapefile)

    # Convert coordinates

    gdf.to\_crs(epsg=4326, inplace=True)

    # London

    ladgdf = gdf[gdf['lad11cd'].str.startswith('E090000')]

    return ladgdf

GeoPandas loads the Shapefile, and then we convert the co-ordinates as discussed above, and filter the rows by the LAD code.

Here is simple code to map the LADs. (Code is in census\_geopandas\_script.py.)

import matplotlib.pyplot as plt

from mpl\_toolkits.axes\_grid1 import make\_axes\_locatable  
import census\_read\_data as crd  
import census\_read\_geopandas as crg

# Get LAD GeoPandas DataFrame

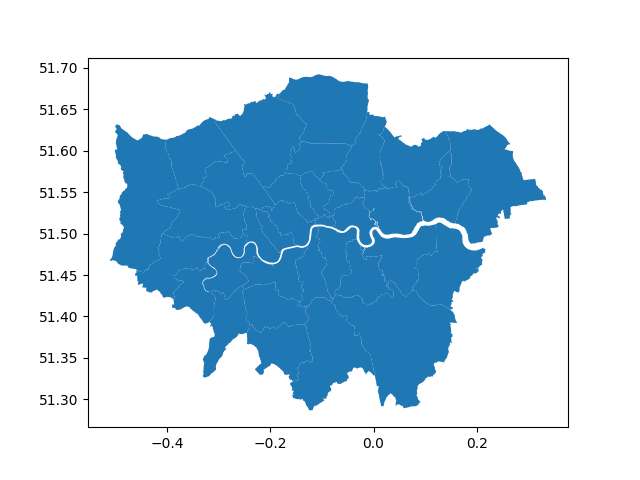
london\_lads\_gdf = crg.read\_london\_lad\_geopandas()

# Default GeoPandas plot

london\_lads\_gdf.plot()

plt.show()

This displays a window with default formatting:



We can load the data to plot on the map as follows:

# Get Census data index and its table\_names

index = crd.read\_index()

table\_names = crd.get\_table\_names(index)

# Get first data table

table\_name = table\_names[0][0]

tdf = crd.read\_table(table\_name)

# Get first row data item (all categories All)

datacol = table\_name + tdf.iloc[0, -1]

# Read the data table (all data items) and merge with the LAD geo data

df = crd.read\_data(table\_name)

gdf = london\_lads\_gdf.merge(df, left\_on='lad11cd', right\_on='GeographyCode')

# Create a Matplotlib plot, turn off axes, position color bar, and plot

fig, ax = plt.subplots(1, 1)

ax.set\_axis\_off()

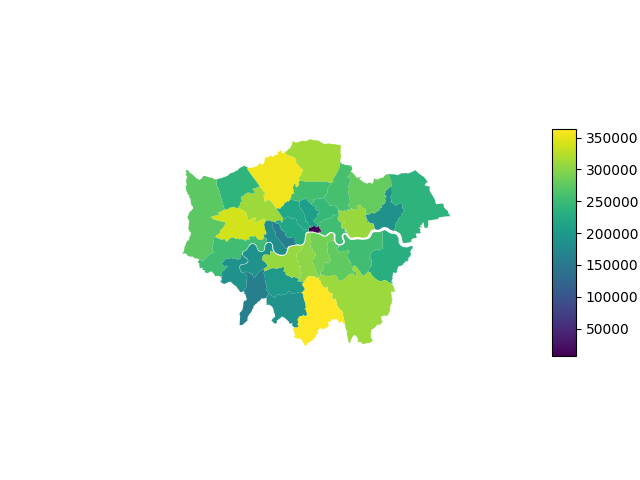
divider = make\_axes\_locatable(ax)

cax = divider.append\_axes("right", size="5%", pad=0.0)

gdf.plot(column=datacol, ax=ax, legend=True, cax=cax)

plt.show()

This displays a window with a choropleth map, colour coded according to the first dataset for the table DC1104EW, i.e. the column DC1104EW0001:



# GeoJSON and Plotly

In order to add interactive features we will use [Plotly](https://plotly.com/python/) and [Dash](https://dash.plotly.com/). Plotly requires geographical data in [GeoJSON](https://en.wikipedia.org/wiki/GeoJSON) format. This format is much larger than Shapefile, and the conversion from Shapefile is time consuming – about 45 seconds for the Ward file on my environment.

I decided to cache the GeoJSON locally. (Code in file census\_read\_geojson.py.)

import geopandas as gpd

import json

import os

def read\_london\_lad\_geojson():

    # Get LAD GeoJSON

    london\_jsonfile = "data/json\_files/London\_LAD\_Boundaries.json"

    if not os.path.exists(london\_jsonfile):

        lad\_jsonfile = "data/json\_files/Local\_Authority\_Districts\_(December\_2011)\_Boundaries\_EW\_BFC.json"

        if not os.path.exists(lad\_jsonfile):

            # Get Census LAD Boundaries as GeoPandas

            shapefile = 'data/Local\_Authority\_Districts\_(December\_2011)\_Boundaries\_EW\_BFC/Local\_Authority\_Districts\_(December\_2011)\_Boundaries\_EW\_BFC.shp'

            ladgdf = gpd.read\_file(shapefile)

            # Convert coordinates

            ladgdf.to\_crs(epsg=4326, inplace=True)

            # Write GeoJSON

            ladgdf.to\_file(lad\_jsonfile, driver='GeoJSON')

        with open(lad\_jsonfile) as f:

            census\_lads = json.load(f)

        london\_lads = census\_lads

        london\_lads['features'] = list(filter(

            lambda f: f['properties']['lad11cd'].startswith('E090000'), london\_lads['features']))

        with open(london\_jsonfile, 'w') as f:

            json.dump(london\_lads, f)

    else:

        with open(london\_jsonfile) as f:

            london\_lads = json.load(f)

    return london\_lads

There are two cached JSON files: the London LAD Boundaries, and the complete LAD Boundaries. The JSON files are simply converted from GeoPandas.

The first plotly map uses the GeoJSON files. (Code in census\_plotly\_script.py.)

import census\_read\_data as crd

import census\_read\_geojson as crg

import pandas as pd

import plotly.express as px

# Get Census Merged Ward and Local Authority Data

geography = crd.read\_geography()

locationcol = "GeographyCode"

namecol = "Name"

# Get LAD GeoJSON

london\_lads = crg.read\_london\_lad\_geojson()

# Get Census data index and its table\_names

index = crd.read\_index()

table\_names = crd.get\_table\_names(index)

# Get first data table and its categories

table\_name = table\_names[0][0]

tdf = crd.read\_table(table\_name)

# Get first row data item (all categories All)

datacol = table\_name + tdf.iloc[0, -1]

# Read the data table (all data items) and merge with the geography names

df = crd.read\_data(table\_name)

df = pd.merge(df, geography, on=locationcol)

# Filter London Data

london\_lad\_ids = list(map(lambda f: f['properties']

                          ['lad11cd'], london\_lads["features"]))

london\_flags = df[locationcol].isin(london\_lad\_ids)

london\_lad\_df = df[london\_flags]

# Map data by LAD

key = "properties.lad11cd"

max\_value = london\_lad\_df[datacol].max()

title = datacol + " by Local Authority"

fig = px.choropleth(london\_lad\_df,

                    geojson=london\_lads,

                    locations=locationcol,

                    color=datacol,

                    color\_continuous\_scale="Viridis",

                    range\_color=(0, max\_value),

                    featureidkey=key,

                    scope='europe',

                    hover\_data=[namecol],

                    title=title

                    )

fig.update\_geos(

    fitbounds="locations",

    visible=False,

)

fig.update\_layout(margin=dict(l=0, r=0, b=0, t=30),

                  title\_x=0.5,

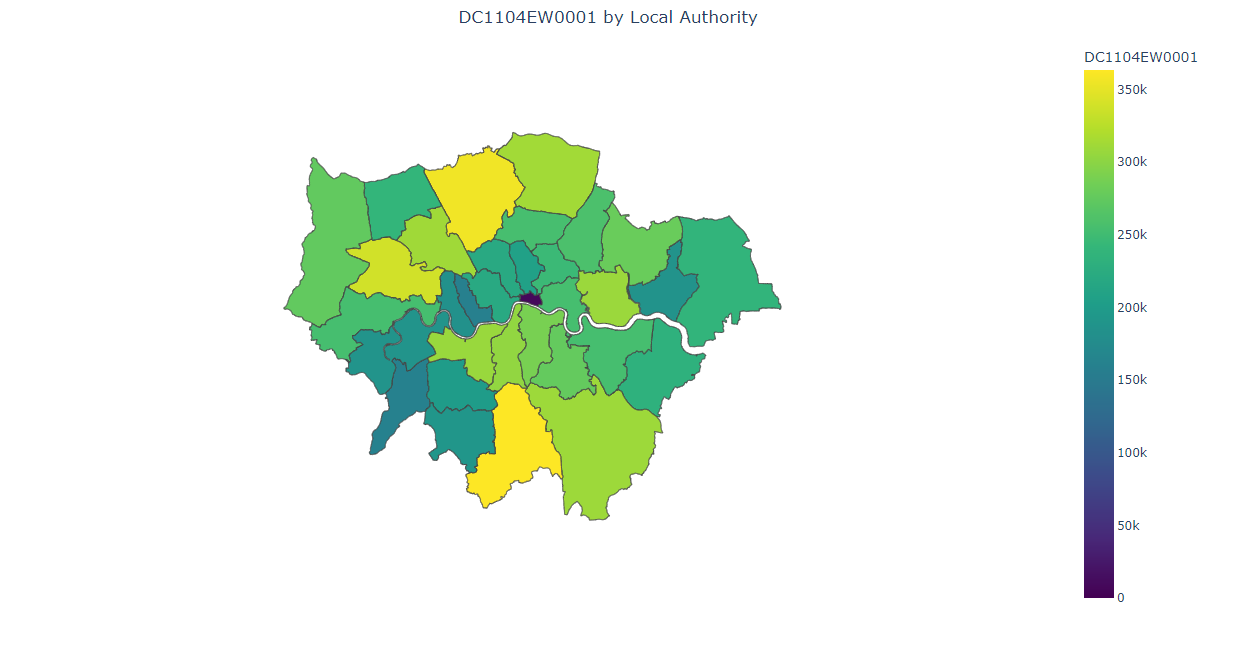
                  width=1200, height=600)

fig.show()

Observations on the code:

* The px.choropleth function call maps the london\_lads GeoJson, colouring the map according to the datacol column of the london\_lad\_df, matching the GeoJson feature property key with the locationcol column.
* The update\_geos function call sets the bounds of the map from the displayed locations.
* The update\_layout function call reduces the margin around the map, and specifies the width and height.

This script creates a browser window:



The code specifies the width and height because a) the default size is smaller, and b) in order to have an appropriate aspect ratio. However, the map is still surrounded by a large amount of white space. I improved this by manually specifying the bounds for the map:

from turfpy.measurement import bbox

from functools import reduce

def compute\_bbox(gj):

    # Compute bounding box for GeoJSON

    gj\_bbox\_list = list(

        map(lambda f: bbox(f['geometry']), gj['features']))

    gj\_bbox = reduce(

        lambda b1, b2: [min(b1[0], b2[0]), min(b1[1], b2[1]),

                        max(b1[2], b2[2]), max(b1[3], b2[3])],

        gj\_bbox\_list)

    return gj\_bbox

gj\_bbox = compute\_bbox(london\_lads)

fig.update\_geos(

    # fitbounds="locations",

    center\_lon=(gj\_bbox[0]+gj\_bbox[2])/2.0,

    center\_lat=(gj\_bbox[1]+gj\_bbox[3])/2.0,

    lonaxis\_range=[gj\_bbox[0], gj\_bbox[2]],

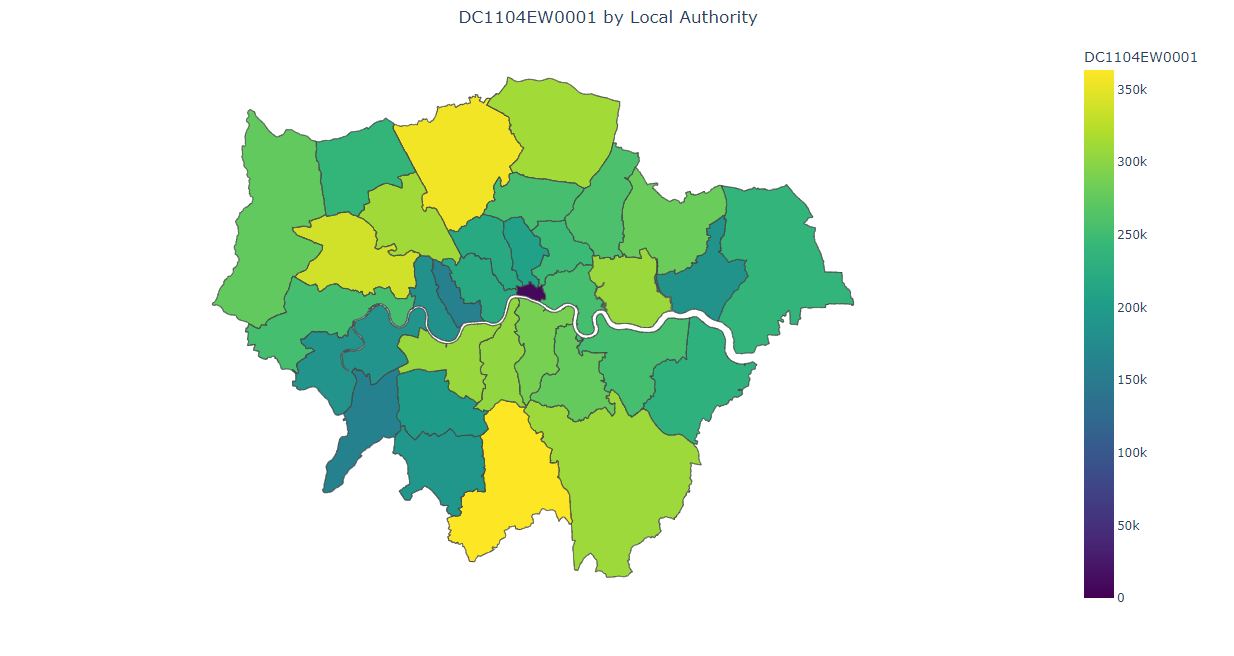
    lataxis\_range=[gj\_bbox[1], gj\_bbox[3]],

    visible=False,

)

Observations on the code:

* The package turfpy.measurement provides a function bbox to compute the bounding box for a GeoJSON feature geometry.
* The function compute\_bbox computes the bounding box for each feature and reduces the list of bounding boxes to compute the combined bounding box.
* Then update\_geos uses the box to specify the center and longitude and latitude ranges for the map.

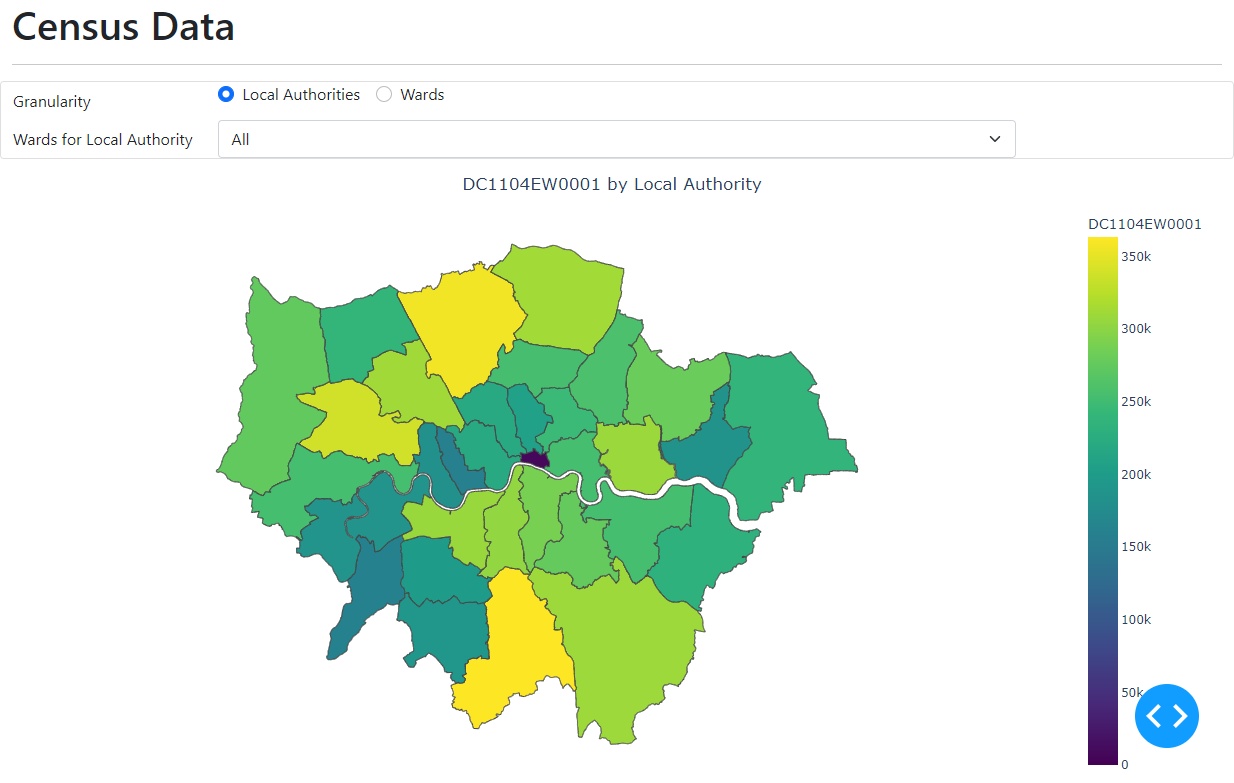


# Dash

[Dash](https://dash.plotly.com/) is built on top of Plotly and “abstracts away all of the technologies and protocols that are required to build a full-stack web app with interactive data visualization”. I used Dash to allow selection of the table, dataset and granularity for the map.

In my first attempt I used the [Dash Core Components](https://dash.plotly.com/dash-core-components) to add the selection controls. While functional, the appearance was not great, so I switched to using [Dash Bootstrap Components](https://dash-bootstrap-components.opensource.faculty.ai/), which provide the consistent Bootstrap look and feel to the controls without needing CSS expertise.

The first version just allows selection of the map granularity, adding these controls:



The Granularity radio items change the granularity of the map between Local Authority, which we have seen so far, and Ward, for a more detailed map. The dropdown selection optionally specifies the Local Authority for a Ward map.

The code is in the file census\_dash\_script\_simple.py.

import census\_read\_data as crd

import census\_read\_geojson as crg

import pandas as pd

import plotly.express as px

import plotly.graph\_objects as go

from dash.dependencies import Input, Output

from dash import html

from dash import dcc

import dash

import dash\_bootstrap\_components as dbc

from turfpy.measurement import bbox

from functools import reduce

# Get Census Merged Ward and Local Authority Data

geography = crd.read\_geography()

locationcol = "GeographyCode"

namecol = "Name"

# Get London GeoJSON

london\_wards = crg.read\_london\_ward\_geojson()

# Get LAD GeoJSON

london\_lads = crg.read\_london\_lad\_geojson()

# Get Census data index and its table\_names

index = crd.read\_index()

table\_names = crd.get\_table\_names(index)

# Get first data table and its categories

table\_name = table\_names[0][0]

tdf = crd.read\_table(table\_name)

# Get first row data item (all categories All)

datacol = table\_name + tdf.iloc[0, -1]

# Read the data table (all data items) and merge with the geography names

df = crd.read\_data(table\_name)

df = pd.merge(df, geography, on=locationcol)

# Filter London Data

london\_ward\_ids = list(map(lambda f: f['properties']

                           ['cmwd11cd'], london\_wards["features"]))

london\_flags = df[locationcol].isin(london\_ward\_ids)

ldf = df[london\_flags]

london\_lad\_ids = list(map(lambda f: f['properties']

                          ['lad11cd'], london\_lads["features"]))

london\_flags = df[locationcol].isin(london\_lad\_ids)

london\_lad\_df = df[london\_flags]

ward\_max\_value = ldf[datacol].max()

lad\_max\_value = london\_lad\_df[datacol].max()

# Dash

def blank\_fig():

    # Blank figure for initial Dash display

    fig = go.Figure(go.Scatter(x=[], y=[]))

    fig.update\_layout(template=None)

    fig.update\_xaxes(showgrid=False, showticklabels=False, zeroline=False)

    fig.update\_yaxes(showgrid=False, showticklabels=False, zeroline=False)

    return fig

def compute\_bbox(gj):

    # Compute bounding box for GeoJSON

    gj\_bbox\_list = list(

        map(lambda f: bbox(f['geometry']), gj['features']))

    gj\_bbox = reduce(

        lambda b1, b2: [min(b1[0], b2[0]), min(b1[1], b2[1]),

                        max(b1[2], b2[2]), max(b1[3], b2[3])],

        gj\_bbox\_list)

    return gj\_bbox

app = dash.Dash(\_\_name\_\_, external\_stylesheets=[dbc.themes.BOOTSTRAP])

local\_authorities = london\_lad\_ids

all\_local\_authorities = ['All'] + local\_authorities

map\_controls = dbc.Card(

    [

        dbc.Row([

            dbc.Label("Granularity", html\_for="granularity", width=2),

            dbc.Col(

                [

                    dbc.RadioItems(

                        id='granularity',

                        options=[{'label': i, 'value': i}

                                 for i in ['Local Authorities', 'Wards']],

                        value='Local Authorities',

                        inline=True

                    ),

                ],

                width=8

            )

        ]),

        dbc.Row([

            dbc.Label('Wards for Local Authority',

                      html\_for="local-authority", width=2),

            dbc.Col(

                [

                    dbc.Select(

                        id='local-authority',

                        options=[

                            #{'label': i, 'value': i}

                            {'label': 'All' if i == 'All'

                                else geography[geography[locationcol] == i][namecol].iat[0],

                                'value': i}

                            for i in all\_local\_authorities],

                        value='All'

                    )

                ],

                width=8

            )

        ]),

    ]

)

app.layout = dbc.Container(

    [

        html.H1("Census Data"),

        html.Hr(),

        dbc.Col(

            [

                dbc.Row(map\_controls),

                dbc.Row(dcc.Graph(id='map', figure=blank\_fig()),

                        class\_name='mt-3'),

            ],

            align="center",

        ),

    ],

    fluid=True,

)

@app.callback(

    Output('map', 'figure'),

    Input('local-authority', 'value'),

    Input('granularity', 'value'),

)

def update\_graph(local\_authority, granularity):

    if granularity == 'Local Authorities':

        fdf = london\_lad\_df

        gj = london\_lads

        key = "properties.lad11cd"

        max\_value = lad\_max\_value

        title = datacol + " by Local Authority"

    else:

        key = "properties.cmwd11cd"

        max\_value = ward\_max\_value

        if local\_authority == 'All':

            fdf = ldf

            gj = london\_wards

            title = datacol + " by Ward"

        else:

            fdf = ldf[ldf['LAD11CD'].str.match(local\_authority)]

            gj = {

                'features': list(filter(lambda f: f['properties']['lad11cd'] == local\_authority,

                                        london\_wards["features"])),

                'type': london\_wards['type'],

                'crs': london\_wards['crs']

            }

            title = datacol + " by Ward for Local Authority"

    gj\_bbox = compute\_bbox(gj)

    fig = px.choropleth(fdf,

                        geojson=gj,

                        locations=locationcol,

                        color=datacol,

                        color\_continuous\_scale="Viridis",

                        range\_color=(0, max\_value),

                        featureidkey=key,

                        scope='europe',

                        hover\_data=[namecol, 'LAD11NM'],

                        title=title

                        )

    fig.update\_geos(

        center\_lon=(gj\_bbox[0]+gj\_bbox[2])/2.0,

        center\_lat=(gj\_bbox[1]+gj\_bbox[3])/2.0,

        lonaxis\_range=[gj\_bbox[0], gj\_bbox[2]],

        lataxis\_range=[gj\_bbox[1], gj\_bbox[3]],

        visible=False,

    )

    fig.update\_layout(margin=dict(l=0, r=0, b=0, t=30),

                      title\_x=0.5,

                      width=1200, height=600)

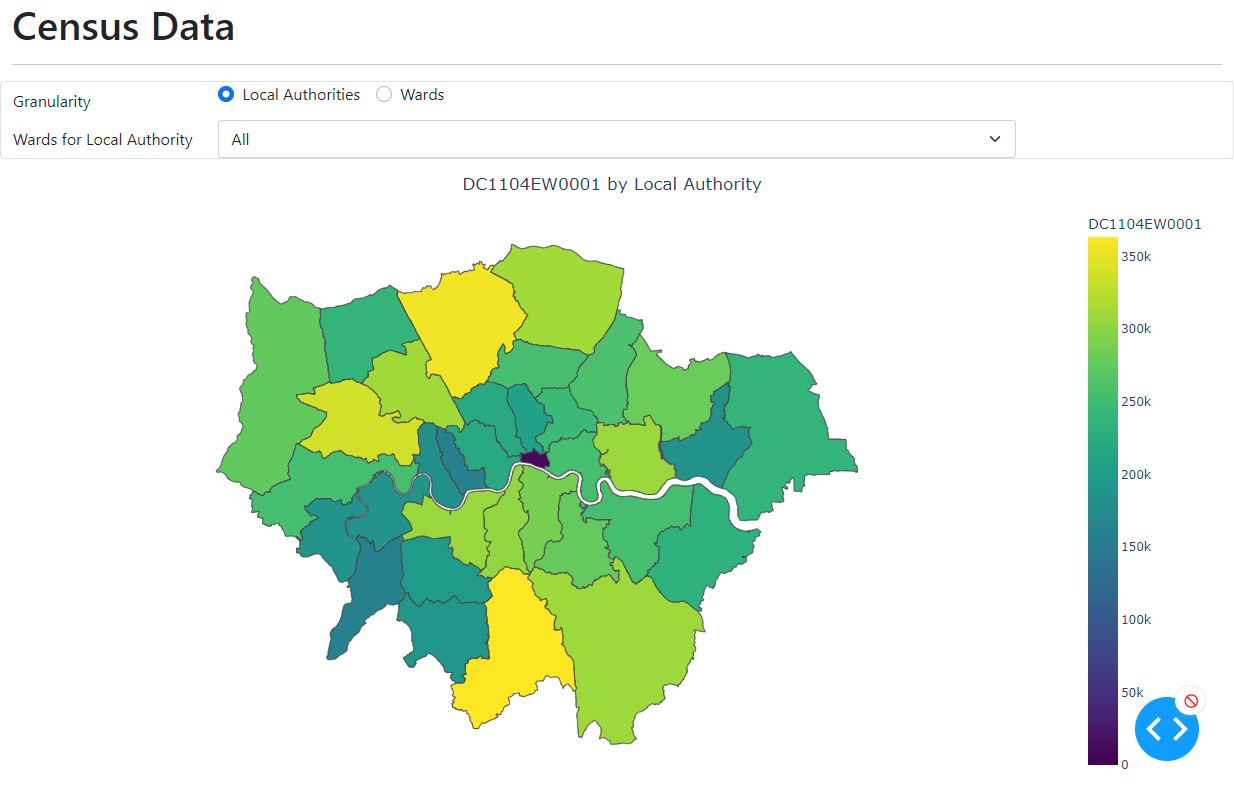
    return fig

if \_\_name\_\_ == '\_\_main\_\_':

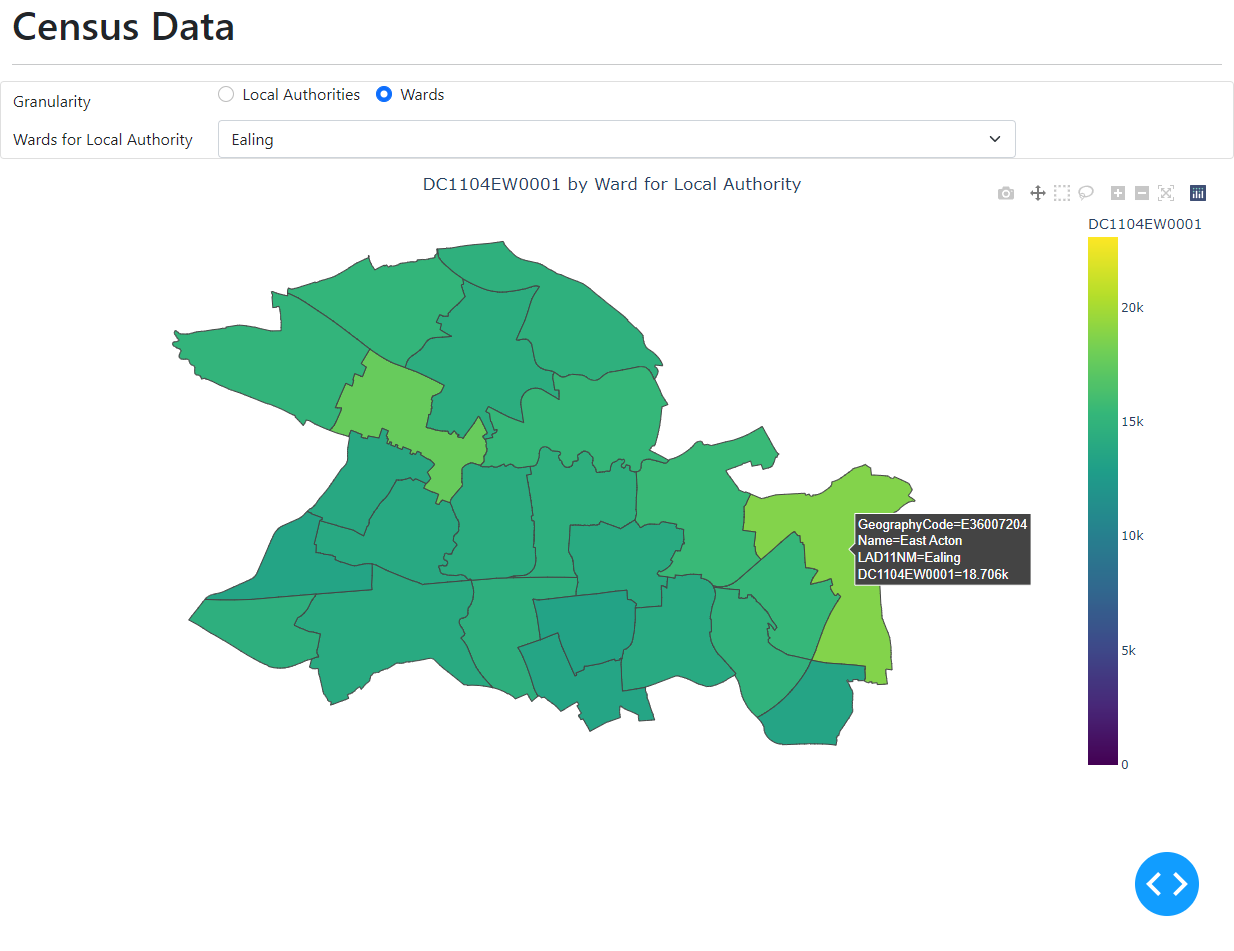
    app.run\_server(debug=True)

Observations on the code:

* This script reads the Ward data using crd.read\_london\_ward\_geojson() in addition to the LAD data.
* The Dash functionality starts after the comment #Dash.
* The initial figure displayed by Dash is a chart, which I did not want to see, so the function blank\_fig() creates a blank figure to use as the initial display. (Credit to this [Stack Overflow answer](https://stackoverflow.com/a/66738747/6721549).)
* The call to dash.Dash() creates the Dash application, using the standard Bootstrap stylesheet.
* The assignment to map\_controls creates the selection controls.
* The assignment to app.layout creates a container for the page with heading, map\_controls and placeholder for the map initially showing the blank figure.
* The @app.callback annotation defines a callback function that is called when either of the selection controls is updated, returning the updated figure.
* The callback builds the map, using Plotly as before, using the data appropriate to the controls. One minor change is the hover\_data=[namecol, 'LAD11NM'], which adds the LAD name to the hover display.
* app.run\_server() starts the Dash server. In my environment it is accessed on <http://127.0.0.1:8050/> and displays this page:



If I change the granularity to Ward and select the Local Authority *Ealing* I get this map; the image shows the hover text with the LAD name:

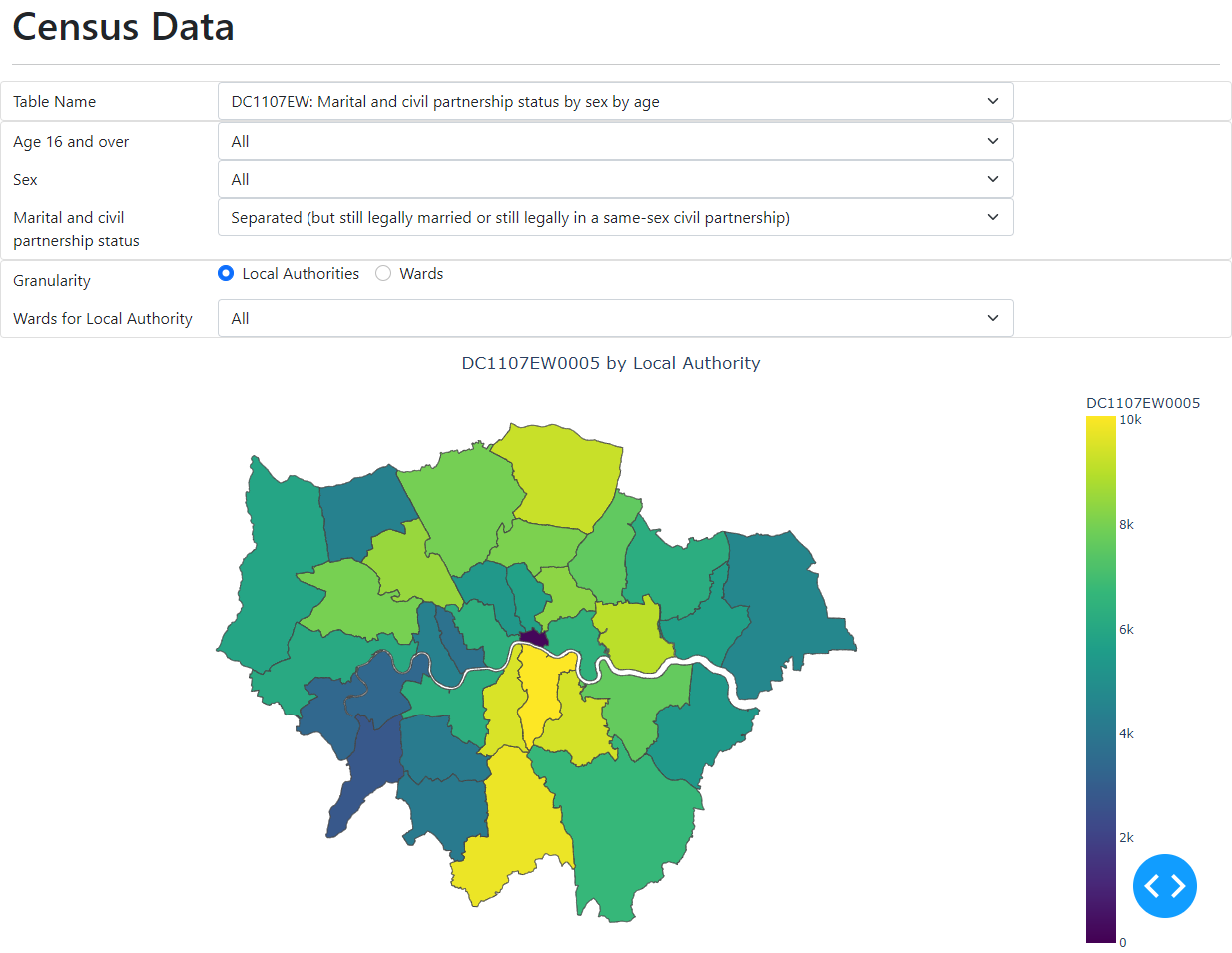


Next I added controls to select the table and dataset. This required many more inputs and outputs on the callback. Ideally, I would like to have multiple callbacks chained together as described in [Dash Basic Callbacks](https://dash.plotly.com/basic-callbacks). However, callbacks must be stateless, so it is not possible for the table selection to update global state with the table data to be displayed on the dataset selection. So, I ended up with one large callback that does everything.

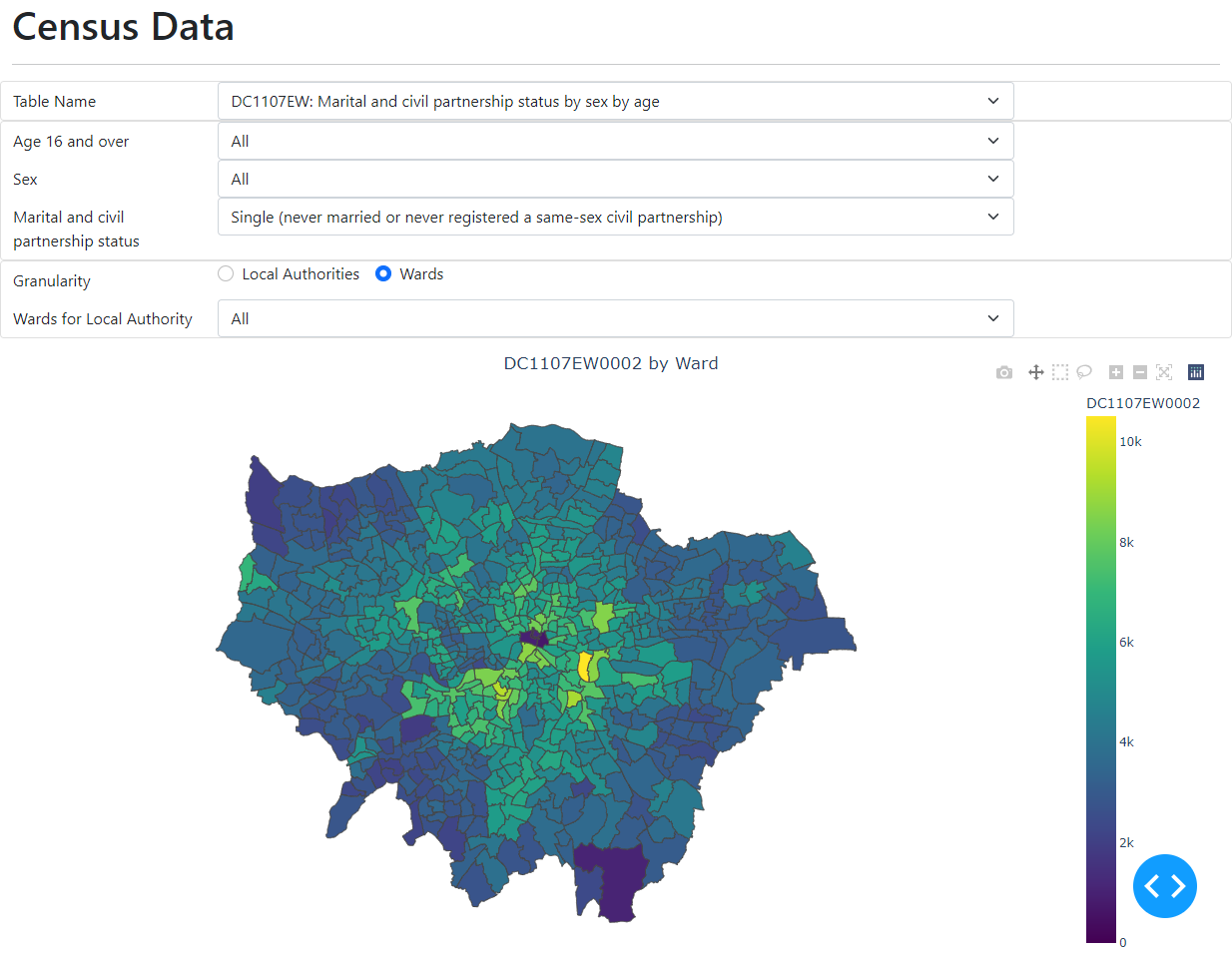
The code is in census\_dash\_script\_full.py. I will not reproduce it here, but summarise the changes:

* The assignment to table\_controls creates the table selection control.
* The assignment to category\_controls creates four category selection controls. Initially they are empty and the style is set to display none so they are invisible.
* The app.layout includes the new controls before the map\_controls from above.
* The app.callback defines additional outputs for the category control label, values and style, and additional inputs for the table and category values.
* The callback proceeds as follows:
* If the table\_name is not set then raise the exception PreventUpdate so the figure is not updated
* Call crd.get\_table\_column\_names\_and\_values() to get the category names and values for the table.
* Set the outputs for the category names and values. Also, for categories that have a value already selected build a query string to filter the table by the value.
* If all required categories have selected values then query the table to get the dataset name, load the data file, and display the map for the appropriate column.

A sample display is shown below:



The next display shows Ward level data for all LADs in London:



# Appendix – Python Packages

I used Python 3.9 on Windows 11 and normally use pip to install packages. However, geopandas depends on packages that are implemented in C/C++, so special procedures are required to install it on Windows. (Apparently the install is straightforward on Linux and Mac.)

The article [Using geopandas on Windows](https://geoffboeing.com/2014/09/using-geopandas-windows/) by Geoff Boeing is referenced as the definitive explanation of how to install geopandas on Windows. The comments on the article have many suggestions on how best to proceed. The essential advice is to use [pipwin](https://pypi.org/project/pipwin/) which installs unofficial python package binaries for Windows provided by Christoph Gohlke [here](http://www.lfd.uci.edu/~gohlke/pythonlibs/). These are the steps I followed:

pip install wheel  
pip install pipwin  
pipwin install numpy  
pipwin install pandas   
pipwin install gdal

# Add the new GDAL path to the Windows PATH environment variable, e.g.:  
# C:\Users\<username>\AppData\Local\Programs\Python\Python39\Lib\site-packages\osgeo   
# It seems this must be done before installing Fiona.

pipwin install fiona  
pipwin install pyproj  
pipwin install rtree  
pipwin install shapely  
pipwin install six  
pip install geopandas