interactive maps with folium

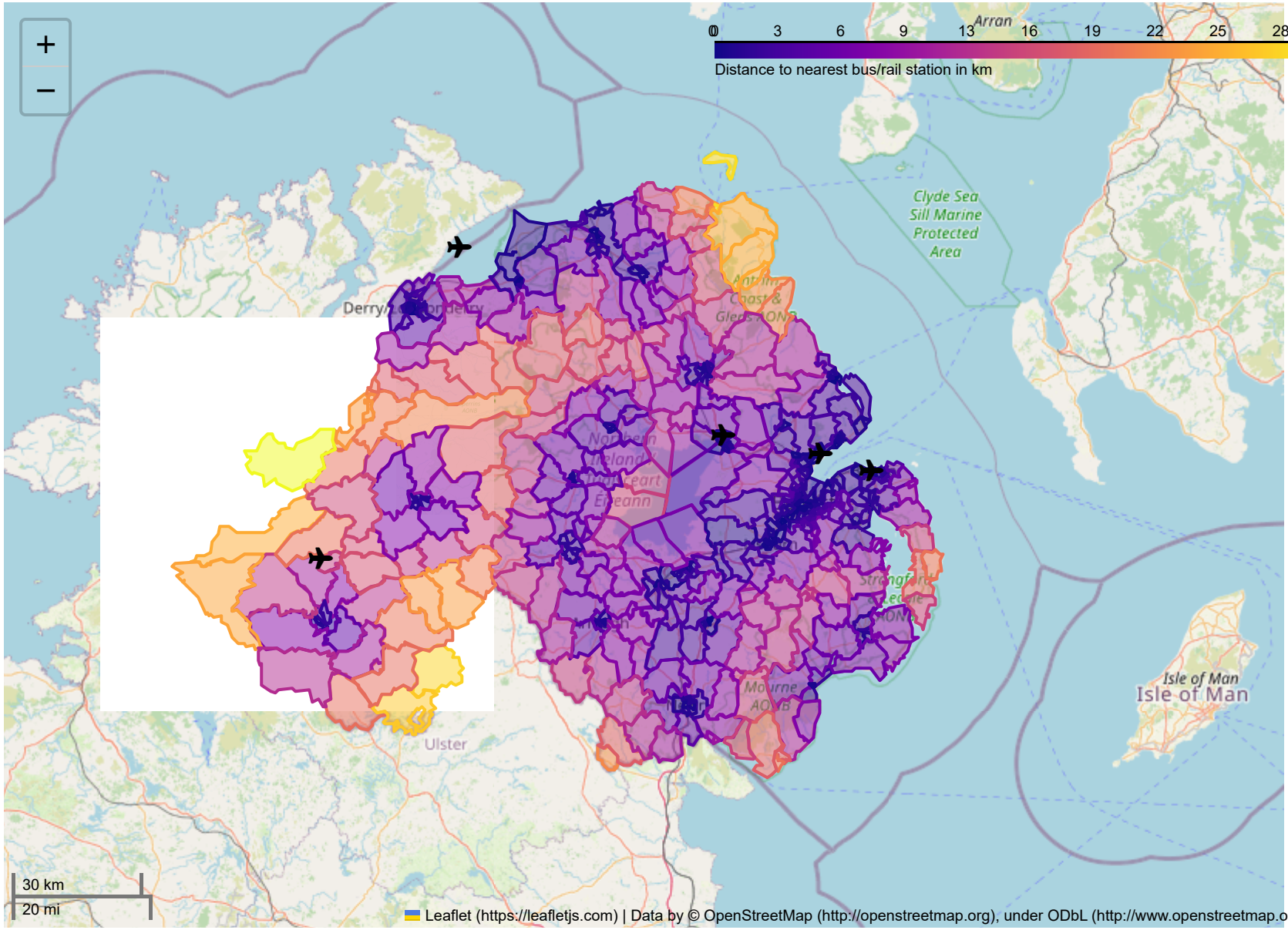
Contents

getting started

In this practical, you'll gain some more experience working with vector data in python using geopandas. You'll see another way to convert text data into geospatial data given coordinates, and you'll see how to join different datasets based on shared attributes.

You will also see how you can create interactive maps using `folium` and `geopandas`, such as the example shown below:

Skip to main content



To begin, launch Jupyter Notebooks as you have in the previous exercises, and begin to work through the notebook (`Week3/Folium.ipynb`)

Skip to main content

> **Note**
>
> Below this point is the **non-interactive** text of the notebook. To actually run the notebook, you'll need to follow the instructions above to open the notebook and run it on your own computer!

Jeremy Leven

overview

In last week's practical, we saw how we can use `cartopy` to create static maps. In the first exercise from this week, we saw how we can use `geopandas` together with `shapely` to work with vector geometries. In this exercise, we'll see how we can use `geopandas`, together with `folium`, to easily create interactive maps.

objectives

- Use `geopandas` to convert csv data with geographic information into vector data
- Use `geopandas.GeoDataFrame.explore()` to create an interactive map from vector data
- Use `pandas.DataFrame.merge()` to join attribute tables based on shared field values

data provided

In the data_files folder, you should have the following files:

Skip to main content

- NI_Wards.shp, a shapefile of electoral wards for Northern Ireland
- transport_data.csv, a csv file with information about public transport for the electoral wards

Note: I will not go over the steps in detail, but if you would like to see how I created the `transport_data.csv` file, the script can be found at **data_files/aggregate_data.py**. To run that script, you will need to download the following files into the **data_files** folder:

- `09-05-2022busstop-list.geojson`, a Geographic JavaScript Object Notation (GeoJSON) file containing locations for all bus stops in Northern Ireland (valid as of May 2022), from OpenDataNI
- `translink-stations-ni.geojson`, a GeoJSON file containing the locations of all Bus and Rail stations in Northern Ireland, from OpenDataNI

getting started

To get started, run the following cell to import the packages we'll use in the exercise.

```
import pandas as pd
import geopandas as gpd
import folium
```

Next, we'll load the Wards dataset using `geopandas`, just like what we saw in the previous exercise:

```
wards = gpd.read_file('data_files/NI_Wards.shp')
```

Remember that we can use `.head()` (documentation) to see a preview of the data:

Skip to main content

```
wards.head()
```

To create an interactive map from the **GeoDataFrame**, we use `.explore()` (documentation), which creates a **folium.Map** (documentation) object.

We'll use the `Population` column to visualize each polygon, and we'll set the color using the viridis colormap from `matplotlib` (more about colormaps):

```
m = wards.explore('Population', cmap='viridis')
```

As you can see, this adds a color scale/legend to the upper right-hand corner of the map, which tells us what the colors of each polygon correspond to. You can zoom in/out to see detail, including on the OpenStreetMap base layer.

And, when you hover over each polygon, you can see additional information about it, taken directly from the attribute table. We could stop here, but in the next sections, we'll see how we can build on this by adding additional data, customizing markers and legend information, and even saving the map to an html file that we can share with others.

converting csv data to vector data, revisited

We've already seen an example of this before in week 1:

```
df = pd.read_csv('data_files/GPSPoints.txt')
df['geometry'] = list(zip(df['lon'], df['lat'])) # zip is an iterator, so we use list to create
                                                 # something that pandas can use.
df['geometry'] = df['geometry'].apply(Point) # using the 'apply' method of the dataframe,
                                             # turn the coordinates column
```

Skip to main content

```
gdf = gpd.GeoDataFrame(df)
gdf.set_crs("EPSG:4326", inplace=True) # this sets the coordinate reference system to epsg:4326, wgs84 lat/lon
```

Here, we'll see how we can use a different method to acheive the same goal.

```
df = pd.read_csv('data_files/Airports.csv') # read the csv data

# create a new geodataframe
airports = gpd.GeoDataFrame(df[['name', 'website']], # use the csv data, but only the name/website columns
                            geometry=gpd.points_from_xy(df['lon'], df['lat']), # set the geometry using points_fro
                            crs='epsg:4326') # set the CRS using a text representation of the EPSG code for WGS84

airports.head() # show the new geodataframe
```

Here, we've used the `geometry` and `crs` arguments of `geopandas.GeoDataFrame.__init__()` (documentation) to do the same thing in a single step.

For the `geometry` argument, we used `gpd.points_from_xy()` (documentation) to create the geomtry based on the latitude and longitude information stored in the csv file.

For the `crs` argument, we used the same EPSG code for WGS84 latitude/longitude as before. Now that we have the dataset loaded, we'll see how we can add it to an existing `folium.Map` object.

adding data to an existing map

In the documentation for `.explore()`, you might notice the following argument:

Skip to main content

```
Existing map instance on which to draw the plot.
```

Earlier, we used the default option of `None`, which created a new map object. Since we already have a map object in place, we can pass this as an argument to add additional data to the map.

We also have a few additional arguments here - we'll say more about what those do after the jump.

```
# add the airport points to the existing map
airports.explore('name',
                 m=m, # add the markers to the same map we just created
                 marker_type='marker', # use a marker for the points, instead of a circle
                 popup=True, # show the information as a popup when we click on the marker
                 legend=False, # don't show a separate legend for the point layer
                )
```

As you can see, the default color for the **Marker** style is blue with a white circle in the middle. Later on, we will see how we can customize the marker to use other colors and icons.

joining tables based on attribute data

In the previous exercise, we saw how we can use a *spatial join* to combine vector data based on their spatial relationship. Sometimes, though, we will need to combine data that have spatial information with data that don't have spatial information - in that case, we'll need to *join* the tables based on some shared attribute.

To see how this works, we can first load the information about public transportation for each electoral ward using `pandas`:

Skip to main content

```
transport.head()
```

Note that this dataset doesn't have any geospatial information, not even latitude/longitude coordinates. But, it does have a Ward Code attribute, which matches the Ward Code attributes from the `wards` shapefile.

Because these attributes are shared between the two tables, we can use `geopandas.GeoDataFrame.merge()` (documentation) to perform a *join* operation.

```
merged = wards.merge(transport, left_on='Ward Code', right_on='Ward Code')
merged.head()
```

To join the two tables, we use the `left_on` and `right_on` arguments of `merge()`, which tells `merge()` which columns to use from the *left* table, `wards` (what ArcGIS calls the "input table"), and the *right* table, `transport` (what ArcGIS calls the "join table").

customizing legends and markers

Once we have this information, we can create a different map that shows the distance to the nearest bus/rail station (in km) for each electoral ward in Northern Ireland, again using `geopandas.GeoDataFrame.explore()`.

This time, though, we'll make sure to change the legend caption. "Population" is easy enough to understand, but "Distance" probably needs a bit more information - distance to what? In what units? We can use the `legend_kwds` argument to set our own caption:

```
legend_kwds={'caption': 'Distance to nearest bus/rail station in km'} # set the caption to a longer explanation
```

Skip to main content

Note that the form of the `legend_kwds` argument is a **dict** (curly braces, `{` and `}`), with a single key/value pair. There are other arguments that we can pass to the legend, but we'll only set the `caption` for now.

We can also customize the markers for our airport locations - the default is not necessarily informative, as it's not clear what each marker is until we hover over it/click on it.

Here, we pass a **dict** to the `marker_kwds` argument:

```
'marker_kwds': {'icon': folium.Icon(color='red', icon='plane', prefix='fa')} # make the markers red with a plane i
```

The only key/value pair in this **dict** is the `icon`, which tells `folium` how to style the marker. In this case, we want it to be a `folium.Map.Icon` (documentation), with the following arguments:

- `color='red'`
- `icon='plane'`
- `prefix='fa'`

`folium` has support for a number of different icon styles, including FontAwesome and Bootstrap glyphicons. I've creatively chosen the `plane` icon from FontAwesome (`prefix='fa'`) for the airports, and made them `red` to stand out from the background a bit, but feel free to make your own adjustments to this style.

Run the cell below to create the new map:

```
m = merged.explore('Distance', # show the Distance column
                   cmap='plasma', # use the 'plasma' colormap from matplotlib
                   legend_kwds={'caption': 'Distance to nearest bus/rail station in km'} # set the caption to a lo
                  )
```

Skip to main content

```
    'm': m, # add the markers to the same map we just created
    'marker_type': 'marker', # use a marker for the points, instead of a circle
    'popup': True, # show the information as a popup when we click on the marker
    'legend': False, # don't show a separate legend for the point layer
    'marker_kwds': {'icon': folium.Icon(color='red', icon='plane', prefix='fa')} # make the markers red with a pla
}

# use the airport_args with the ** unpacking operator - more on this next week!
airports.explore('name', **airport_args)

m # show the map
```

The last thing we might want to do is save the map to an html file, so that we can share it online:

```
m.save('NI_Transport.html')
```

additional exercises and next steps

That wraps up the introduction to creating interactive maps using `geopandas` and `folium`. If you're looking for additional practice, here are some suggestions to get you started:

- In the `transport` dataset, there is a column called `NumBus`, which corresponds to the number of bus stops in each electoral ward. Use this, and some of the topics covered previously, to create a map that shows the number of bus stops per capita for each electoral ward, rather than the distance to the nearest bus/rail station.
- Download the Translink bus/train station location data from OpenNI, and add these data to the map using a custom marker that shows whether the station is a rail station (`R`), a bus station (`B`), or a mixed-use (`I`) station.

Previous
vector data using shapely and geopandas

Next
raster data using rasterio