# CASA0029: Urban Data Visualisation
# Interactive Mapping Practical

This practical introduces interactive mapping techniques using Leaflet and Mapbox. Leaflet can be used to create simple online maps quickly; while Mapbox is a suite of tools to load large spatial datasets and build complex interactive mapping sites. The aim of this Practical is to provide an overview of techniques that you can use in your own spatial data visualisations. All the interactive mapping software libraries discussed here have good online tutorials where you can explore further mapping examples beyond this practical-

http://leafletjs.com/examples.html

https://leafletjs.com/reference.html

https://www.mapbox.com/help/tutorials/

https://docs.mapbox.com/mapbox-gl-js/example/

https://www.mapbox.com/mapbox-gl-js/api

And for a general HTML and JavaScript reference, a good site is-

https://www.w3schools.com/html/

You will need to use the HTML editor (VS Code) that you installed last week. Also remember that if your HTML code does not work, you can view the error messages in your browser's Developer Tools. In Chrome you go to the menu on the right -> More Tools -> Developer Tools to bring up the console. In Safari you need to turn on the Inspector option.

## Leaflet.js Examples

Go to Moodle and download and unzip the "Practical 2 Data and Examples" file. Open this folder in VS Code to view this week's examples.

Leaflet is a very popular open source library for online interactive maps. It is lightweight and straightforward to use, and is ideal for simpler mapping sites. The API and documentation for Leaflet.js is here-
http://leafletjs.com/

Note that Leaflet began as an offshoot of another open source mapping library called OpenLayers (https://openlayers.org/) which has more features, but is a bit more complicated to use.

Below is a simple html page to create an interactive map with OpenStreetMap data (same example as last week)-

**Leaflet_example1.html**

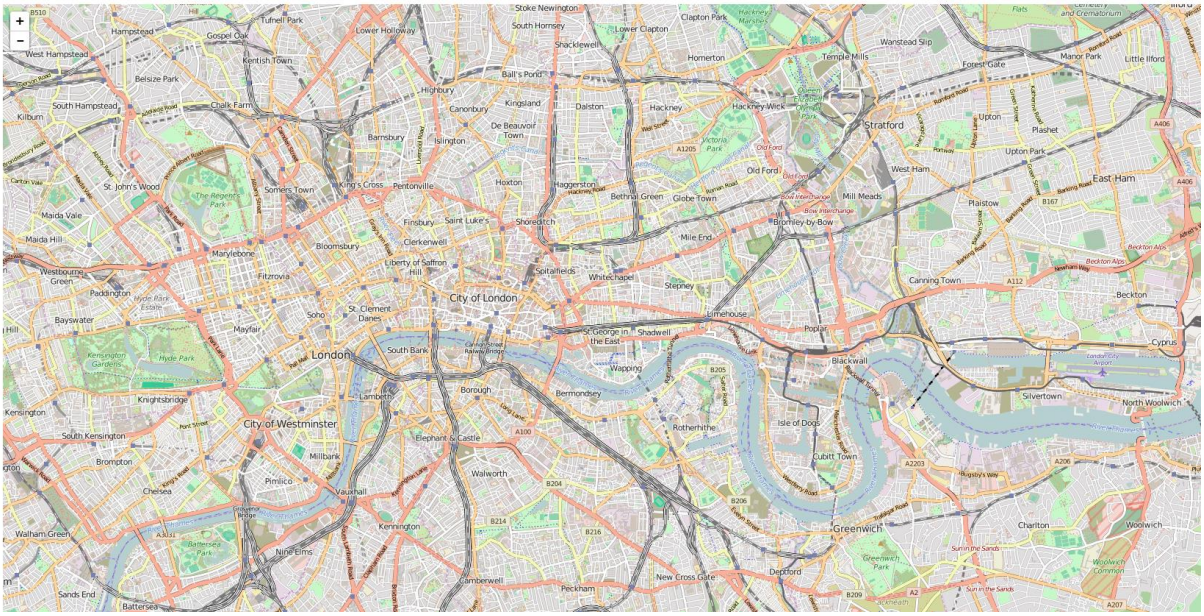
```
<!DOCTYPE html>
<html>
<head>

<title>Leaflet Example Page</title>

 <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY=" crossorigin=""/>

<script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js" integrity="sha256-
20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo=" crossorigin=""></script>


<style>
 body { margin:0; padding:0; }
 #mapdiv { position:absolute; top:0; bottom:0; width:100%; }
</style>

</head>


<body>

<div id="mapdiv"></div>

<script>
    var mymap = L.map('mapdiv').setView([51.505, -0.09], 13);

    L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        maxZoom: 18,
        attribution: 'Map data &copy; <a
href="http://openstreetmap.org">OpenStreetMap</a>'
        }).addTo(mymap);
</script>

</body>

</html>
```

This will produce a simple interactive map with OpenStreetMap data as the background layer, and you can zoom around the map layer will update. The important parts of the code are as follows-

The Leaflet.js library is loaded in the <head> part of the page, using the <link> tag to load the Leaflet stylesheet, and the <script> tag to load the Leaflet JavaScript content of the library. The stylesheet controls the layout of different elements, and ensures the map and map controls appear in the correct position on the page.

The map itself is created in the <body> part of the page using a  tag to specify the size and location of the map element, and a <script> that contains the Leaflet commands to create the map. An instance of the L.map class is invoked, and the setView() method is used to specify the latitude, longitude and zoom level of the map. Then the 'L.tileLayer' class is used to create the background

layer, linking to a set of tiles on an online map server. This is an example of a raster map tiles layer stored on a Map Tile Server, as discussed during the lecture. The raster tiles contain OpenStreetMap data, and are added to the map using the addToMap() method.




Note that in the code we are combining a variable declaration, instantiating the L.map class, and running the setView method in the following statement-

```
var mymap = L.map('mapdiv').setView([51.505, -0.09], 13);
```

This statement can alternatively be split into two lines, performing the same tasks-

```
var mymap = L.map('mapdiv');

mymap.setView([51.505, -0.09], 13);
```

Similarly the tileLayer statement is also combining a class instantiation and running the addToMap() method in the same statement. We could instead perform the same task using the following code-

```
var myTileLayer = L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
{maxZoom: 18, attribution: 'Map data &copy; <a
href="http://openstreetmap.org">OpenStreetMap</a>' });

myTileLayer.addTo(mymap);
```

It's possible to swap the tileLayer for another raster map layer that you want to use, for example a Google Maps layer-

**Leaflet_example3.html (extract)**

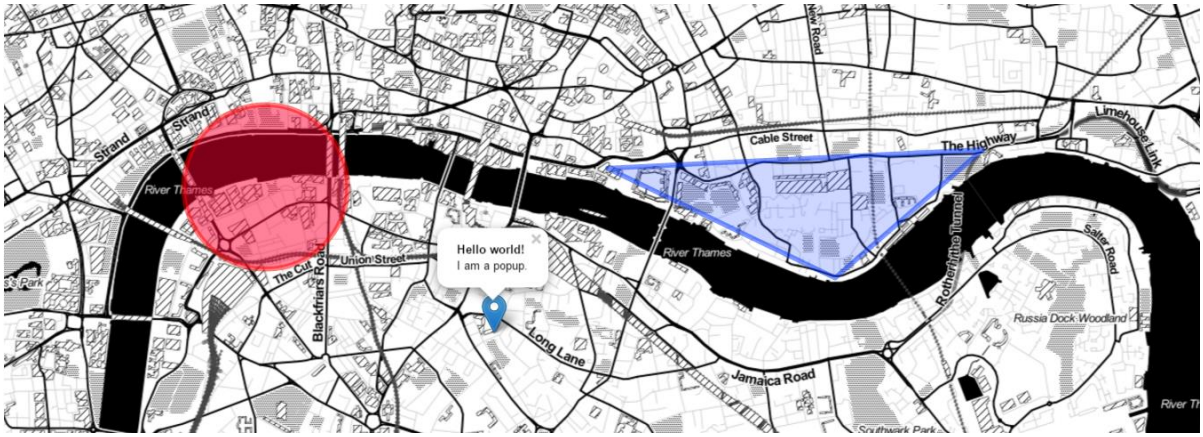
```
...
  L.tileLayer('https://mt1.google.com/vt/lyrs=r&x={x}&y={y}&z={z}', {
        maxZoom: 18,
        attribution: 'Map data &copy; <a href="https://maps.google.com/">Google</a>'
        }).addTo(mymap);
...
```

## Adding Simple Data to a Leaflet Map

So far, we have created the background layer to our map. Typically, we want to add some location data in the foreground. The simplest method of doing this is to hand-code location data in our script, as the example below shows using the commands L.marker, L.circle and L.polygon. Note the new methods bindPopup and openPopup are used to add popup interactivity to these features-

**Leaflet_example4.html (extract)**

```
…
<body>

<div id="mapdiv"></div>
<script>

var mymap = L.map('mapdiv').setView([51.505, -0.09], 13);

   L.tileLayer('https://mt1.google.com/vt/lyrs=r&x={x}&y={y}&z={z}', {
            maxZoom: 18,
            attribution: 'Map data &copy; <a href="https://maps.google.com/">Google</a>'
            }).addTo(mymap);


L.marker([51.5, -0.09]).addTo(mymap)
         .bindPopup("<b>Hello world!</b><br />I am a popup.").openPopup();

L.circle([51.508, -0.11], 500, {
         color: 'red',
         fillColor: '#f03',
         fillOpacity: 0.5
}).addTo(mymap).bindPopup("I am a circle.");

L.polygon([
         [51.509, -0.08],
         [51.503, -0.06],
         [51.51, -0.047]
]).addTo(mymap).bindPopup("I am a polygon.");

</script>

</body> …
```




Hand-coding location features is a very limited approach to visualising spatial datasets. For larger datasets, we would typically either upload the data to a server (this is the approach we are going to use with Mapbox in this tutorial); use a flat file such as a GeoJSON or CSV (we will use this method next week); or we would request a subset of a large dataset using an online API (we will discuss this approach in Week 5).

## Mapbox Introduction

The Leaflet.js examples demonstrate that it is straightforward to create basic interactive maps. There are however limitations with this approach: firstly, we cannot create our own custom background map style with Leaflet; and secondly hand coding spatial data or importing via APIs limits us to smaller spatial datasets.

In the next series of examples, we use Mapbox, a cloud based mapping tool with powerful cartographic tools and a WebGL based JavaScript client library. We will create custom background maps, and then host a vector tile layer which we will use to create an interactive map where the user can change the data year and view a popup of the source data. The first example will be an interactive map of London Underground passenger numbers. Then we will also look at a choropleth map of UK house prices.

Mapbox is really a suite of several integrated tools. A good overview of the all the Mapbox tools can be found here- https://docs.mapbox.com/help/getting-started/ . In this example we are going to use-

**Mapbox Studio**- Online map editing tool where you can create your own basemaps as vector tiles. OpenStreetMap data has been preloaded, and you can also upload your own data (tilesets). Finished maps are called 'Styles'.

**Mapbox Tileset-** Tilesets are spatial datasets (shapefiles, GeoJSON) that have been converted into vector tiles (a scalable vector format for spatial data) for online mapping. You can upload your own spatial data layers which Mapbox then converts into Tilesets to use in your Mapbox maps. Tilesets can be used in both Mapbox Studio and in Mapbox.gl maps.

**Mapbox.gl JS**- A client-side JavaScript mapping library similar to Leaflet but using WebGL technology. You can create html pages that show your Mapbox Studio styles, and add Mapbox Tilesets directly and style them using JavaScript on the client side. This latter approach allows you to create dynamic maps that can be changed with user interaction.

## Create a Free Mapbox Account

You should be able to go to https://www.mapbox.com/ and create a free account. Skip the step that asks for credit card information. If you are unable to create an account, there is a CASA Student account available. The details for this account will be available on the Slack forum.

## Mapbox.gl JS Example 1

The first example shows the structure for adding the Mapbox.gl library to an HTML page, very similar to the Leaflet example. In the page header, we add the Mapbox external stylesheet and script links. In the page body we create a map div, then add a script that creates a new mapbox.gl map and assigns this to the map div.

One difference with Mapbox is that in includes an 'access token'. This is a basic security feature to limit external access to your data that you upload to Mapbox. In the example below an access token is provided. You should replace this with your own access token, which can be found in the Tokens section of your online Mapbox account.

**Mapbox_example1.html**

```
<!DOCTYPE html>
<html>
<head>
<title>Mapbox Example 1 - Load a Basic Map</title>
<meta name="viewport" content="initial-scale=1,maximum-scale=1,user-scalable=no">

<link href="https://api.mapbox.com/mapbox-gl-js/v3.18.0/mapbox-gl.css"
rel="stylesheet">  // Add mapbox.gl stylesheet
<script src="https://api.mapbox.com/mapbox-gl-js/v3.18.0/mapbox-gl.js"></script>  // Add
mapbox.gl JavaScript library

<style>
body { margin: 0; padding: 0; }
#map { position: absolute; top: 0; bottom: 0; width: 100%; }
</style>

</head>


<body>
<div id="map"></div>

<script>
mapboxgl.accessToken =
'pk.eyJ1IjoiZHVuY2FuMjAwMSIsImEiOiIyRDhtOE1nIn0.OA5QmCprkPbOoxZog8HIow';  //Put your
mapbox access token here

const map = new mapboxgl.Map({
   container: 'map', // container ID
   center: [-0.1, 51.5], // starting position [lng, lat]
   style: 'mapbox://styles/mapbox/standard', // Basemap style. Can be changed, e.g.
standard-satellite
   zoom: 9 // starting zoom
});

map.addControl(new mapboxgl.NavigationControl());

</script>


</body>
</html>
```

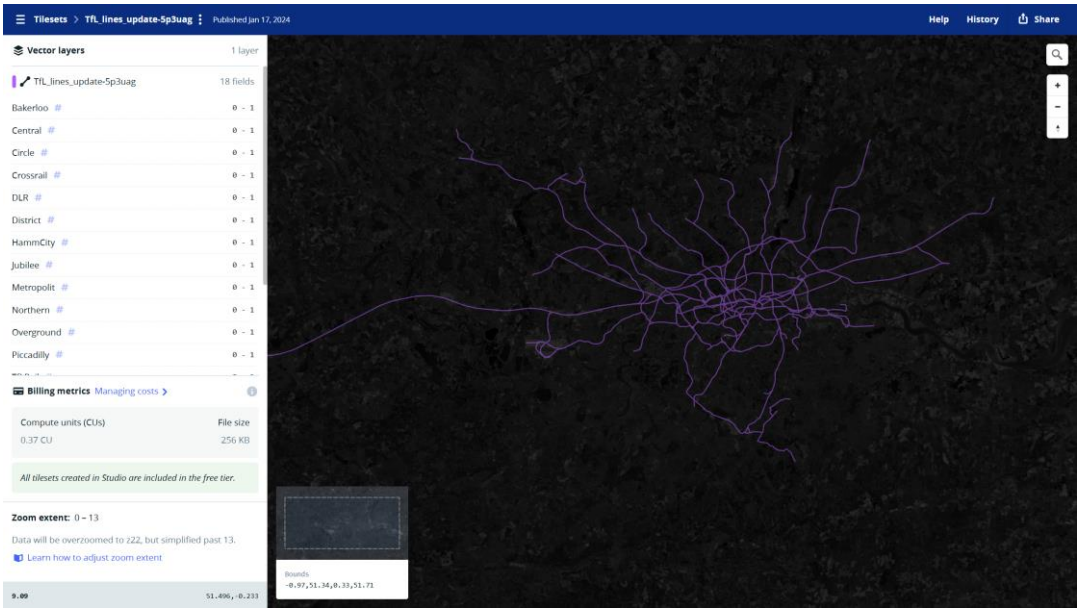

Mapbox Example 1 shows the default Mapbox vector tiles style. If you look closely you can see that the map geometry data is vector, compared to the raster tiles we used in the Leaflet example.

## Creating a Mapbox Tileset

We are going to use Mapbox to visualise our own custom data. Our first example will map how busy London Underground stations are. In the Practical 2 Data and Examples folder, there are two Tube datasets: "TfL_lines_update.zip" and "TubeEntryExit.zip". These are both shapefiles that have been zipped up. The source of this data can be found here- http://crowding.data.tfl.gov.uk/

And the source of the London Underground Tube Lines and Station Location data is from Ollie O'Brien's website here-
https://github.com/oobrien/vis

Go to Mapbox.com (you will need to create a free account if you haven't already), login and click on the 'Data Manager' button on the top left. Then click on 'New Tileset', and 'Select a File'. Select the "TfL_lines_update.zip" file. This will then be uploaded to Mapbox and a Tileset will be created. Click on the new Tileset to view it. You should be able to see the geometry of the Tube lines (zoom in from the world view) and the data columns included in this file-



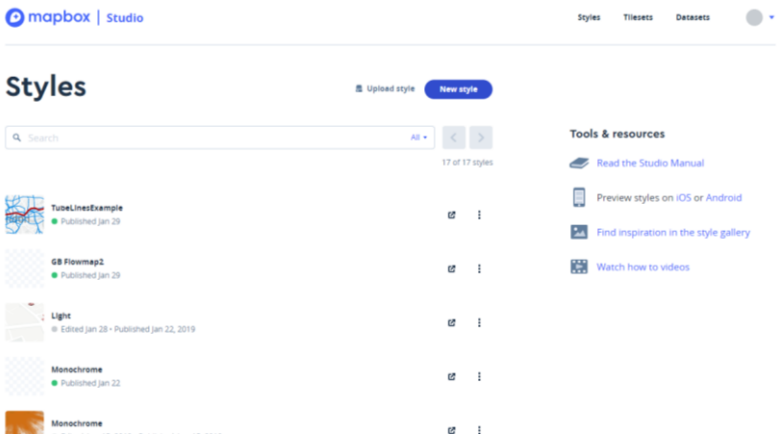

This Tileset page has important information such as the Map ID (press the "Share" button, top right) and the layer name (which are used to show this layer in an HTML page using mapbox.gl). It also has the zoom extent of the layer (which zoom levels the data will be visible at).

Repeat these steps and create another Tileset for the "TubeEntryExit.zip" file, which contains the tube station locations and their total number of annual entries and exits.

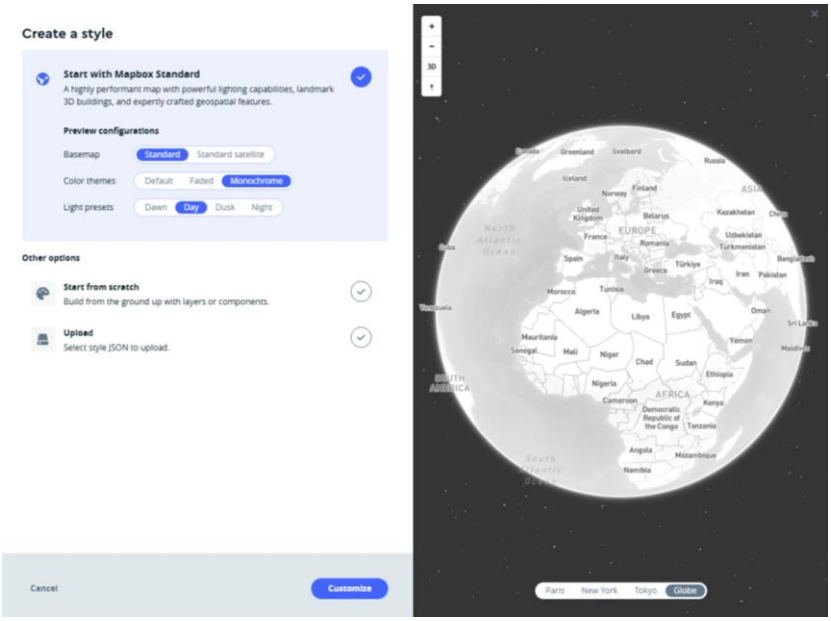
## Designing a Tube Line Basemap in Mapbox Studio

The basic structure for most interactive maps is that you have a data layer in the foreground with the main map content, and a basemap in the background showing spatial context. It is often useful to create a custom basemap that highlights only the relevant information for the map you are displaying. In this case we are going to create a custom basemap of London tube lines.

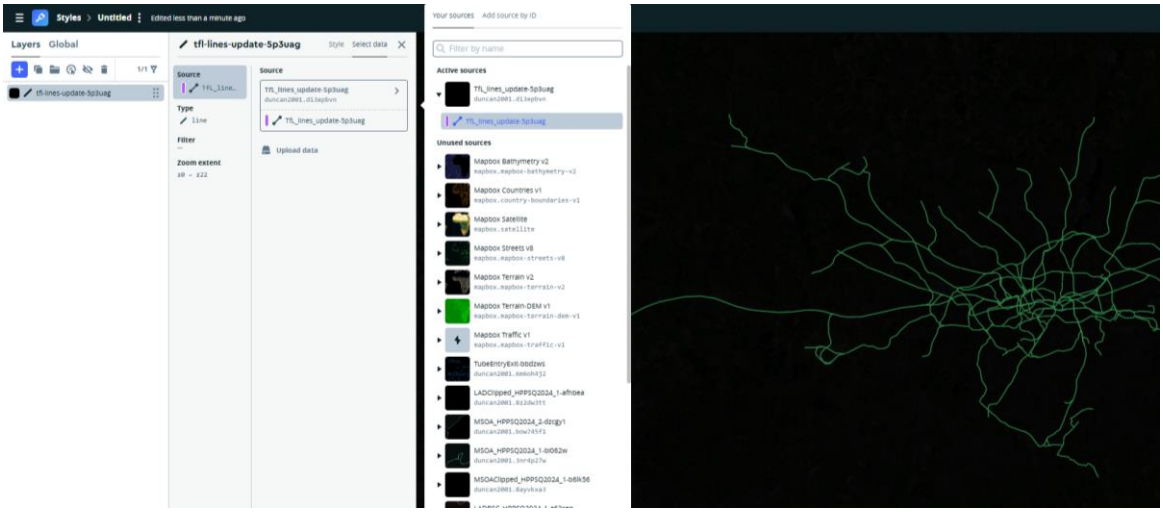
Basemap designs are called Styles in Mapbox. Click on the Style Editor button at the top-left of the page. Then the 'New Style' button-



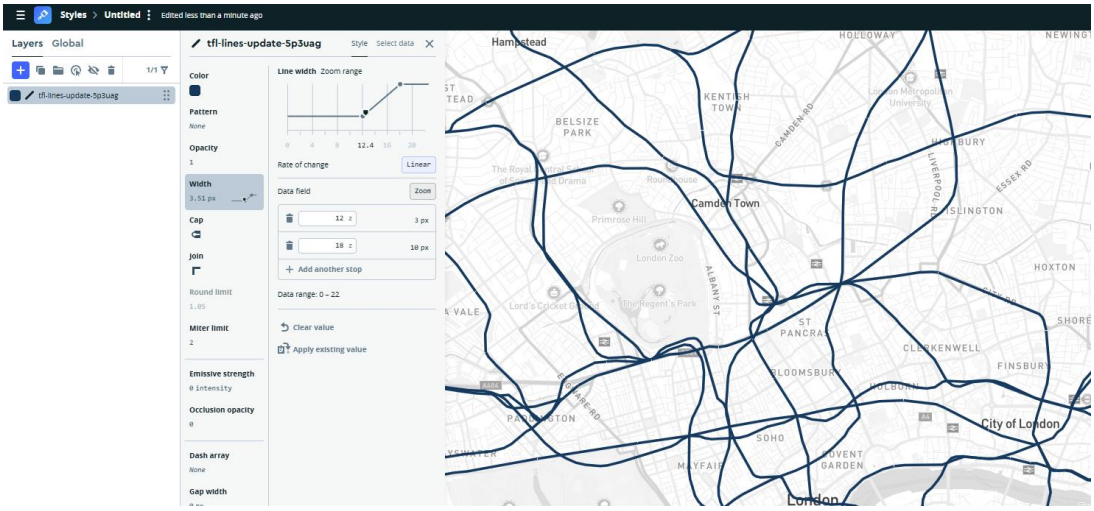

There are a wide selection of style templates to choose from, mainly using OpenStreetMap (OSM) data. It is interesting to explore these different styles for different projects, with the minimalist 'Dark' and 'Light' styles generally the default for data visualisations. For this example, we want to use the 'Monochrome- Day' style-




Press the "Customize" button and you will then be taken to the Mapbox Studio interface for designing basemaps. Layers are on the left-hand side, similar to GIS software. Navigate the map to centre on the UK. Click on the "+" button at the top left to add a new custom layer. We want to add the TfL Lines Tileset that we created earlier. You should be able to select this Tileset from the list of sources. You will have to click on the lines layer as shown below-



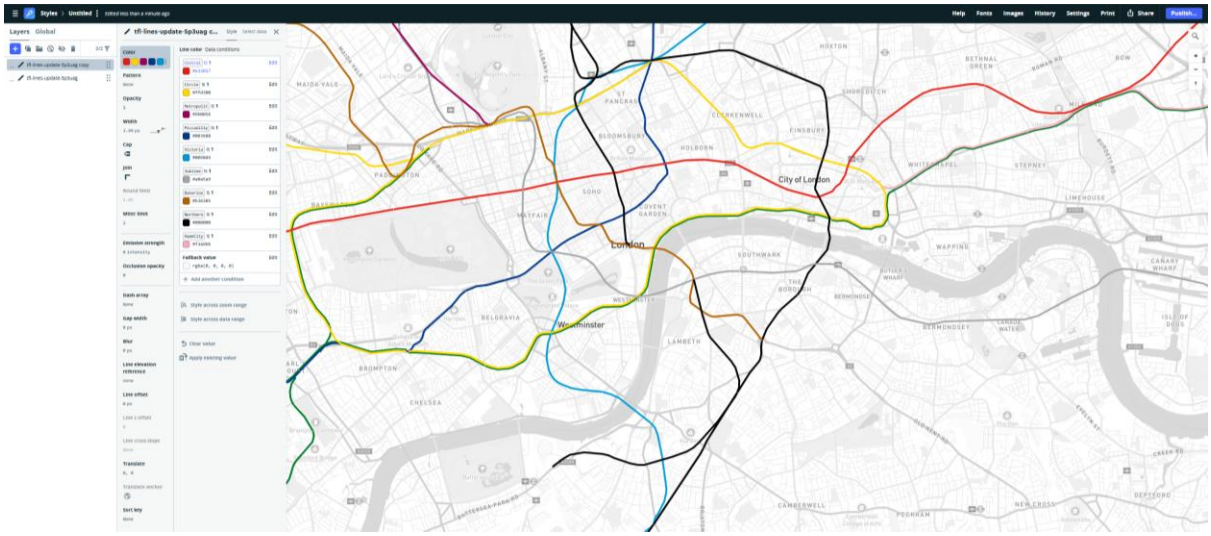
When added, the lines layer will appear on the left-hand layers panel. Next, we are going to style this layer. Click on the style tab at the top (next to Select Data). Firstly, change the colour to something bold such as red or dark blue. Next, we want to set the width of the lines. Often, we want map style settings to change with zoom levels. To achieve this in Mapbox Studio, press the Style Across Zoom Range button. You can then adjust the line settings to become thicker as the user zooms in-



You can recreate a style similar to the official London Underground Map by colouring each line with the TfL colours, though this takes a while to achieve. To do this, you need to style the tube line colours according to a data condition. This allows you to set a different colour for each tube line. Note the dataset we are using does not have data for Elizabeth Line, DLR or Overground stations.

Another way to achieve the same outcome is to filter each layer (in Source Data) for each line. You can duplicate layers and copy styles by right clicking the layers in the panel. The filter method allows you to offset lines that are overlapping (such as the Circle and District lines).

The full documentation for Mapbox Studio can be found here-
https://www.mapbox.com/help/studio-manual/

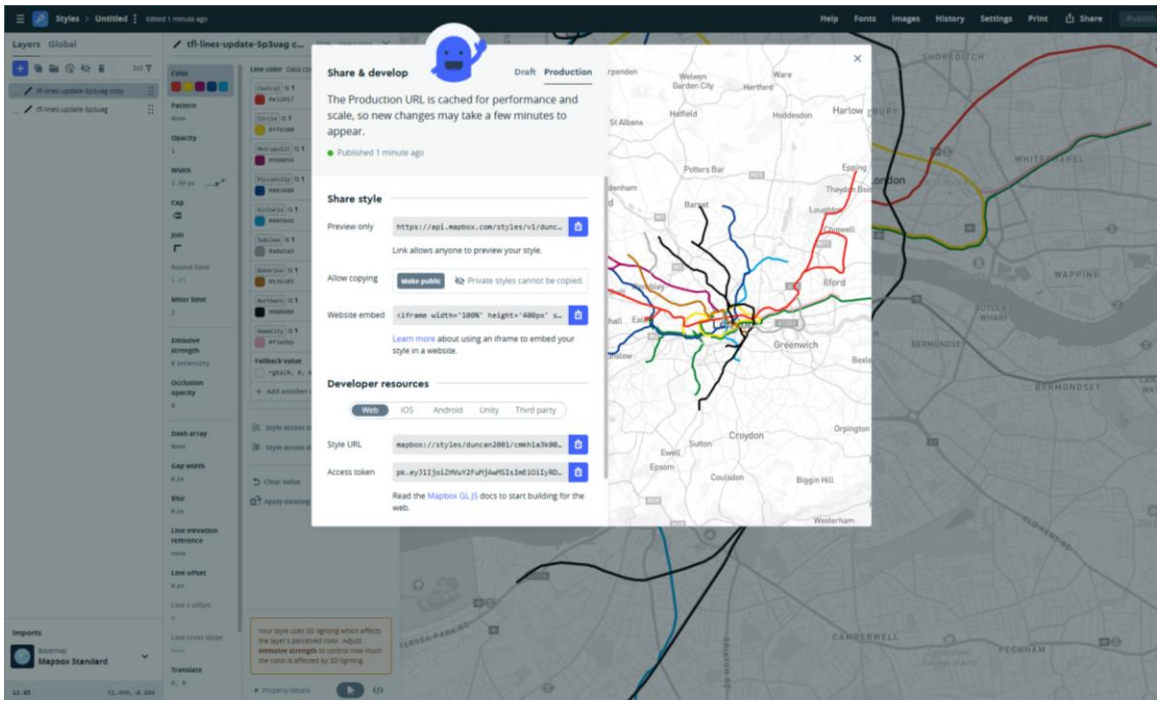



The official colours in hex format are shown below-

| Line | Hex | RGB |
|---|---|---|
| Bakerloo | #B36305 | 179, 99, 5 |
| Central | #E32017 | 227, 32, 23 |
| Circle | #FFD300 | 255, 211, 0 |
| District | #00782A | 0, 120, 42 |
| Elizabeth* | #6950a1 | 105, 80, 161 |
| Hammersmith & City | #F3A9BB | 243, 169, 187 |
| Jubilee | #A0A5A9 | 160, 165, 169 |
| Metropolitan | #9B0056 | 155, 0, 86 |
| Northern | #000000 | 0, 0, 0 |
| Piccadilly | #003688 | 0, 54, 136 |
| Victoria | #0098D4 | 0, 152, 212 |
| Waterloo & City | #95CDBA | 149, 205, 186 |
| DLR | #00A4A7 | 0, 164, 167 |
| London Overground | #EE7C0E | 238, 124, 14 |
| London Trams | #84B817 | 132, 184, 23 |

## Publishing Your Tube Map Style

Rename your style into something more descriptive than ‘Untitled’, for example ‘London Tube Lines’. The ‘Publish’ button at the top right will save your style edits. Then press the ‘Share’ button-




The Share button provides the details to include this style in your own Mapbox.gl pages. You will need the Style URL and your Access Token. This is shown in ‘Mapbox_example2_TubeMap.html’ in the Moodle example files. Change the access token and style ID to your own layer as shown below-

**Mapbox_example2_TubeMap.html (extract)**

```
…

<body>
<div id="map"></div>

<script>
mapboxgl.accessToken =
'pk.eyJ1IjoiZHVuY2FuMjAwMSIsImEiOiIyRDhtOE1nIn0.OA5QmCprkPbOoxZog8HIow';  //Put
your mapbox access token here

// Load a new map in the 'map' HTML div
var map = new mapboxgl.Map({
 container: 'map', // container id
 style: 'mapbox://styles/duncan2001/cjb5126wu17g52spxin20qohu', // Style ID for
the London Tube Lines basemap designed in Mapbox Studio
 center: [-0.1, 51.5], // starting position [lng, lat]
 zoom: 12 // starting zoom
});

</script>

</body>
</html>
```

## Interactive Data Example- Underground Stations

Our Tube Lines map has navigational interactivity only- the user can pan and zoom, but cannot change the data layers or styling of the map. We will now add a proportional circle layer of London Underground Annual Entry and Exit Passengers that we added as a Tileset earlier.

### Mapbox GL JavaScript Library

Mapbox.gl JS is the main client-side JavaScript library for viewing maps created using Mapbox tools. A major difference with other mapping libraries is that it is based on WebGL, a powerful graphics standard than standard HTML pages. This means that the library can handle more advanced graphics like animations and 3D data. Note you can rotate and tilt Mapbox GL maps by holding down your right mouse button and moving the mouse (this feature can also be turned off on your own maps).

We are going to add the Tube Station data as a new layer on top of our tube line basemap. The Tube Station data is a Tileset, not a style, and we are going to style this Tileset on the client-side in the JavaScript code. This gives us more flexibility to change the style interactively using JavaScript-

**Mapbox_example3_TubeMap.html**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8' />
    <title>Mapbox Example 3 - Add Tube Station Proportional Circles Layer</title>
    <meta name='viewport' content='initial-scale=1,maximum-scale=1,user-scalable=no' />
    <link href="https://api.mapbox.com/mapbox-gl-js/v3.18.0/mapbox-gl.css"
rel="stylesheet">
    <script src="https://api.mapbox.com/mapbox-gl-js/v3.18.0/mapbox-gl.js"></script>
    <style>
        body { margin:0; padding:0; }
        #map { position:absolute; top:0; bottom:0; width:100%;}
    </style>
</head>
<body>

<div id='map'></div>


<script>
    mapboxgl.accessToken =
'pk.eyJ1IjoiZHVuY2FuMjAwMSIsImEiOiIyRDhtOE1nIn0.OA5QmCprkPbOoxZog8HIow'; // Put your
Mapbox Public Access token here

    // Load a new map in the 'map' HTML div
    var map = new mapboxgl.Map({
    container: 'map', // container id
    style: 'mapbox://styles/duncan2001/cjb5126wu17g52spxin20qohu', // stylesheet
location
    center: [-0.1, 51.5], // starting position [lng, lat]
    zoom: 12 // starting zoom
    });


    map.on('load', function() {    // This is our first example of asynchronous JS. We
can only add additional layers after the map has loaded
          // Add the circle layer to the map

        map.addLayer({
                id: 'TubeEntryExit2', // Name of the layer. Reused in the labels layer
below, as both use same source
                type: 'circle',
```

```
                source: {
                  type: 'vector',
                  url: 'mapbox://duncan2001.6m6oh4j2' // Your Mapbox tileset Map ID
                },
                'source-layer': 'TubeEntryExit-bbdzws', // name of tileset layer
                'layout': {
                    'visibility': 'visible'
                },
                paint: {
                    'circle-color': '#008080',
                    'circle-opacity': 0.95,
                    'circle-radius': {
                        property: 'EE2024', // The data column used for the circle
radius. EE2024 is total annual entries and exits 2024
                        stops: [                              // The circle radius varies
according to the zoom level and the number of passengers
                            [{zoom: 9, value: 0}, 1],
                            [{zoom: 9, value: 100}, 8],
                            [{zoom: 12, value: 0}, 3],
                            [{zoom: 12, value: 100}, 60],
                            [{zoom: 16, value: 0}, 4],
                            [{zoom: 16, value: 100}, 200],
                            ]
                    }}
              });


        map.addLayer({   // Add label layer to the map
            'id': 'labels',
            'type': 'symbol',
            'source': 'TubeEntryExit2', // uses Tileset ID defined the circle layer
above
            'source-layer': 'TubeEntryExit-bbdzws', // name of tilesets
            'layout': {
                'text-field': '{EE2024}m',  // Labels use data variable EE2024
                'text-font': ['Open Sans Bold', 'Arial Unicode MS Bold'],
                'text-size': {
                     stops: [[8, 0], [12.5, 0], [12.6, 12], [16, 20]]
                 }
            },
            'paint': {
                'text-color': 'rgba(255,255,255,0.9)',
                'text-halo-width': 2, // Text halo is a text outline to help visibility
                'text-halo-color': 'rgba(70,70,70,0.6)'
            }
        });

    });

</script>

</body>
</html>
```

The new code is placed inside a function: “map.on('load', function() { … });”. This is an example of an asynchronous JavaScript function- it is run after the background map style has loaded, and is a common technique in JavaScript. Mapbox.gl can only add layers after the main basemap has been loaded on the page. We create two new layers from the TubeEntryExit Tileset. The first is a proportional circle map, where the radius of the circle depends on the ‘EE2008’ field from the original shapefile (this field is the total Entries and Exits in 2008). The “stops” property in Mapbox.gl creates a linear function between zoom levels, just like we used earlier in Mapbox Studio.

The second layer we create from the TubeEntryExit Tileset is a label layer. This provides labels for the total number of entries and exits in 2008, the same variable as is used for the circle sizes. Your map should appear similar to the map below-


## Adding User Interaction and Dynamic Styling to Mapbox

The advantage of styling data using JavaScript on the client-side, rather than preparing our styles in advance in Mapbox Studio, is that we can change the styling dynamically in response to user interaction.

Our final Tube example, which is rather longer, creates an interactive time-slider. When the user moves the time-slider it triggers a function which changes the variable (column in the data table) that the TubeEntryExit layers are displaying, so that the user can change the data year that is shown on the map. This is an event-listener structure, which is common in JavaScript. When a user event occurs (e.g. click on something, mouse-over something) a function is triggered which runs JavaScript code and changes the page.

This final tube map example adds several features to our page-

- a link to an external stylesheet that controls the appearance of the time-slider, the text around the time-slider and the popup.
- a div element 'map-overlay top' within which the map title and time-slider are placed.
- an array variable 'years' that lists the data years used on the time-slider.
- function 'setYear' which is called when the user changes the time-slider. This function changes the paint properties of the TubeEntryExit circle layer, and the TubeEntryExit label layer, and redraws these layers according to the data year that the user has selected.
- a filter for the labels layer than only shows tube values above 30 million passengers.
- an 'event listener' for the time-slider feature that triggers the function 'setYear' every time the time-slider is moved by the user.
- a variable that creates a new Mapbox popup object.

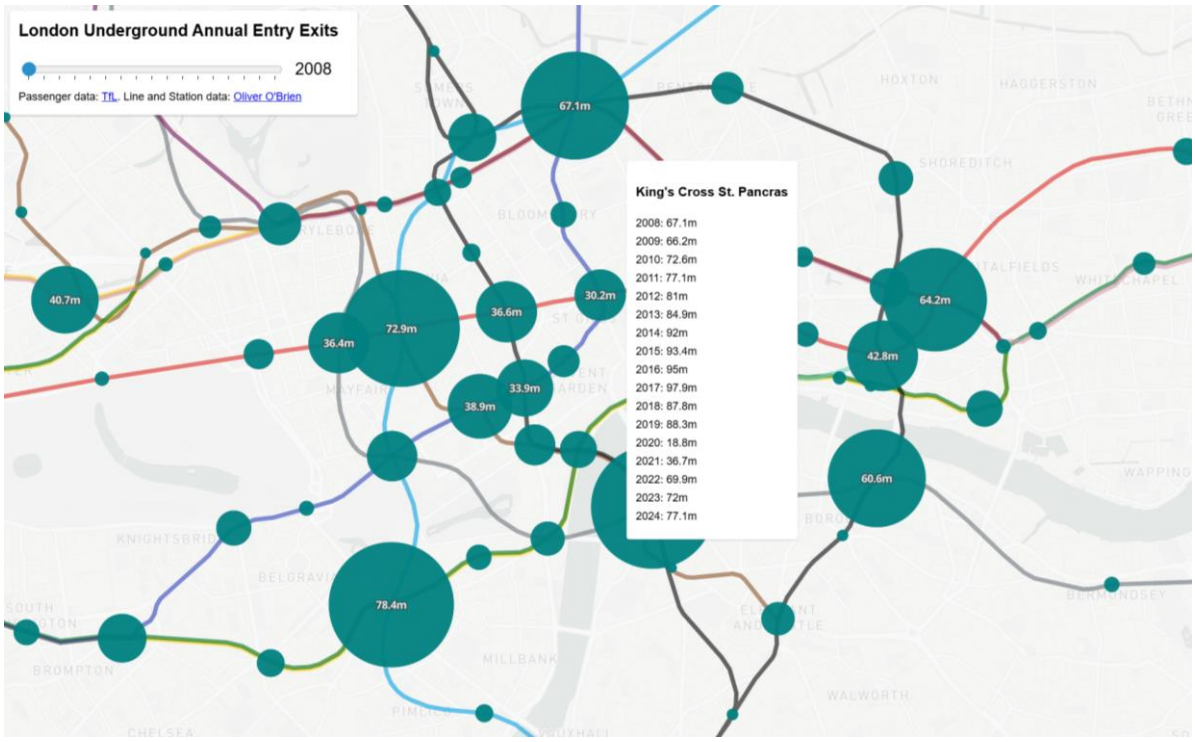
- a ‘mouseover’ function that is triggered when the user puts their pointer over a tube station. The popup then becomes visible.
- a ‘mouseenter’ and a ‘mouseexit’ function that changes the mouse pointer when hovering over a tube station, and removes the popup when moving the mouse off a tube station.

**Mapbox_example4_TubeMap.html**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8' />
    <title>MapBox Example 4 - Add Interactive Timeslider to London Tube Map</title>
    <meta name='viewport' content='initial-scale=1,maximum-scale=1,user-scalable=no' />
    <link href="https://api.mapbox.com/mapbox-gl-js/v3.18.0/mapbox-gl.css" rel="stylesheet">
    <script src="https://api.mapbox.com/mapbox-gl-js/v3.18.0/mapbox-gl.js"></script>
    <link href='MapboxTube_example_styles.css' rel='stylesheet' />
</head>

<body>

<div id='map'></div>



<div class='map-overlay top'>
    <div class='map-overlay-inner'>
        <h2>London Underground Annual Entry Exits</h2>
        <table><tr><td>
        <input id='slider' type='range' min='0' max='16' step='1' value='0' list='tickmarks' />  <!-- Create timeslider with values
from 0 to 16, corresponding to the data years -->
        <datalist id="tickmarks">
          <option value="0" label="2008">
          <option value="1">
          <option value="2">
          <option value="3">
          <option value="4">
          <option value="5">
          <option value="6" label="2014">
          <option value="7">
          <option value="8">
          <option value="9">
          <option value="10">
          <option value="11">
          <option value="12">
          <option value="13">
          <option value="14">
          <option value="15">
          <option value="16" label="2024">
          </datalist>
         </td>
         <td>
          <label id='year'>2008</label>
         </td>
         </tr></table>
         <p class="credit">Passenger data: <a href="http://crowding.data.tfl.gov.uk/">TfL</a>. Line and Station data: <a
href="https://github.com/oobrien/vis/tree/master/tube/data">Oliver O'Brien</a></p>
</div>
</div>



<script>
    mapboxgl.accessToken = 'pk.eyJ1IjoiZHVuY2FuMjAwMSIsImEiOiIyRDhtOE1nIn0.OA5QmCprkPbOoxZog8HIow'; // Put your Mapbox Public Access
token here

    // Load a new map in the 'map' HTML div
    var map = new mapboxgl.Map({
    container: 'map', // container id
    style: 'mapbox://styles/duncan2001/cjb5126wu17g52spxin20qohu', // style location
    center: [-0.1, 51.5], // starting position [lng, lat]
    zoom: 12 // starting zoom
    });



    // Create an array of the available data years. Array positions are equivalent to the timeslider values
    var years = [
        '2008',
        '2009',
        '2010',
        '2011',
        '2012',
        '2013',
        '2014',
        '2015',
        '2016',
        '2017',
        '2018',
        '2019',
        '2020',
        '2021',
        '2022',
```

```
        '2023',
        '2024'
    ];

    // Add zoom and rotation controls to the map.
    map.addControl(new mapboxgl.NavigationControl());

    map.on('load', function() {


            // This is the main function that runs when the user changes the data year timeslider. The function input paramater is
the timeslider position variable year
            function setYear(year) {

                document.getElementById('year').textContent = years[year];  // Set the label to the correct year using the years
array

                var pp = map.getPaintProperty('TubeEntryExit2','circle-radius');  // Paint property will allow circle radius to be
changed

                console.log(pp);
                pp.property = "EE" + years[year];  // update the paint property variable circle-radius to the new column set by the
user

                map.setPaintProperty('TubeEntryExit2','circle-radius',pp);  // Change the circle radius with the new column using the
setPaintProperty method

                console.log(map.getPaintProperty('TubeEntryExit2','circle-radius'));

                var yearcol = "EE" + String(years[year]);
                var textfield = "{" + yearcol + "}m";

                console.log(textfield);

                map.setLayoutProperty('labels', 'text-field', textfield); // update the labels layer to the new column
                var filters = ['>', yearcol, 30];
                map.setFilter('labels', filters);
            }


          // Add the proportional circle layer to the map
            map.addLayer({
                    id: 'TubeEntryExit2',
                    type: 'circle',
                    source: {
                      type: 'vector',
                      url: 'mapbox://duncan2001.6m6oh4j2' // Your Mapbox tileset Map ID
                    },
                    'source-layer': 'TubeEntryExit-bbdzws', // name of tileset
               'layout': {
                    'visibility': 'visible'
                },
                paint: {
                    'circle-color': '#008080',
                    'circle-opacity': 0.95,
                    'circle-radius': {
                       property: 'EE2008',
                       stops: [                              // The circle radius varies according to the zoom level and the number
of passengers
                           [{zoom: 9, value: 0}, 1],
                           [{zoom: 9, value: 100}, 8],
                           [{zoom: 12, value: 0}, 3],
                           [{zoom: 12, value: 100}, 60],
                           [{zoom: 16, value: 0}, 4],
                           [{zoom: 16, value: 100}, 200],
                           ]
                    }}
                 });

            // Add the label layer to the map
        map.addLayer({
            'id': 'labels',
            'type': 'symbol',
            'source': 'TubeEntryExit2',
            'source-layer': 'TubeEntryExit-bbdzws', // name of tilesets
            'layout': {
                'text-field': '{EE2008}m',
                'text-font': ['Open Sans Bold', 'Arial Unicode MS Bold'],
                'text-size': {
                     stops: [[8, 0], [12.5, 0], [12.6, 12], [16, 20]]
                 }
            },
            'paint': {
                'text-color': 'rgba(255,255,255,0.9)',
                'text-halo-width': 2,
                'text-halo-color': 'rgba(70,70,70,0.6)'
            }
        });

            var filters = ['>', 'EE2008', 30];  // Only show the labels for the larger tube stations using a filter
            map.setFilter('labels', filters);

            var prevyear = 0;

            // Assign an event listner to the slider so that the setYear function runs when the user changes the slider
            document.getElementById('slider').addEventListener('input', function(e) {
                var year = parseInt(e.target.value);
```

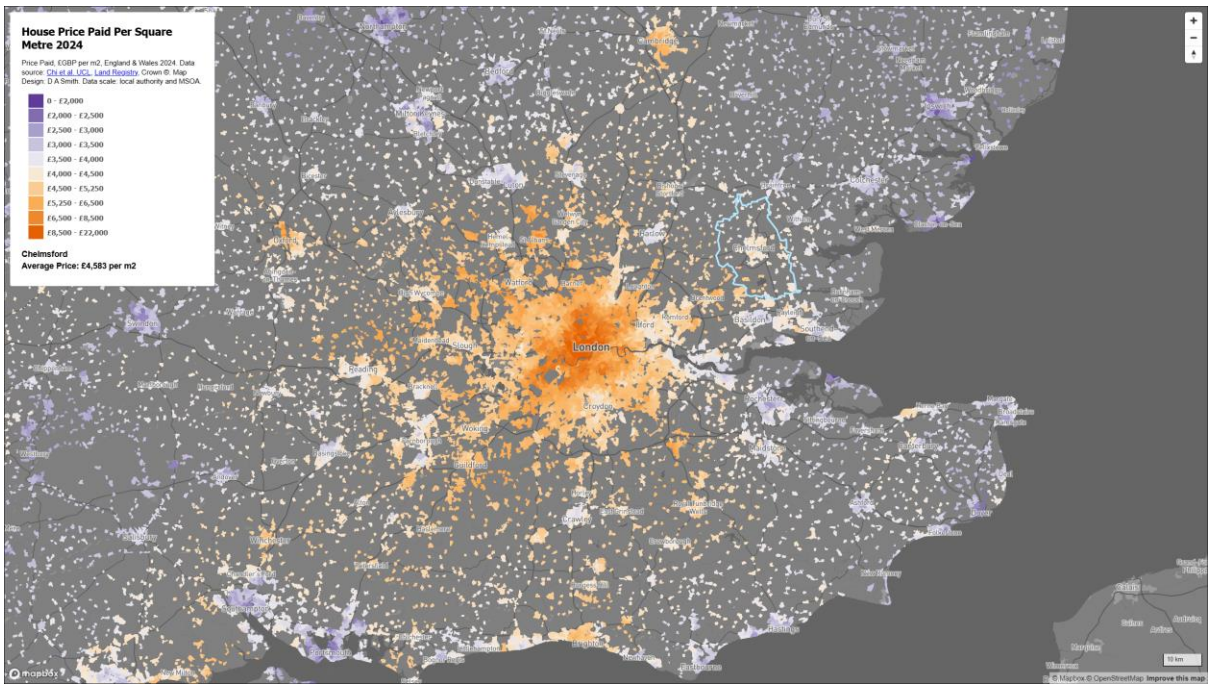
```
                setYear(year);  // Run function setYear with the slider position as the input
                });


            var mypopup = new mapboxgl.Popup({offset:[150,50],closeButton: false});

            // Another event listener that adds the popup when the user puts their cursor over the tube circles
            map.on('mouseover', 'TubeEntryExit2', function (e) {
                    mypopup
                        .setLngLat(e.features[0].geometry.coordinates)
                        .setHTML("<h3>" + e.features[0].properties.name + "</h3>2008: " + e.features[0].properties.EE2008 + "m<br
/>2009: " + e.features[0].properties.EE2009 + "m<br />2010: " + e.features[0].properties.EE2010 + "m<br />2011: " +
e.features[0].properties.EE2011 + "m<br />2012: " + e.features[0].properties.EE2012 + "m<br />2013: " +
e.features[0].properties.EE2013 + "m<br />2014: " + e.features[0].properties.EE2014 + "m<br />2015: " +
e.features[0].properties.EE2015 + "m<br />2016: " + e.features[0].properties.EE2016 + "m<br />2017: " +
e.features[0].properties.EE2017 + "m<br />2018: " + e.features[0].properties.EE2018 + "m<br />2019: " +
e.features[0].properties.EE2019 + "m<br />2020: " + e.features[0].properties.EE2020 + "m<br />2021: " +
e.features[0].properties.EE2021 + "m<br />2022: " + e.features[0].properties.EE2022 + "m<br />2023: " +
e.features[0].properties.EE2023 + "m<br />2024: " + e.features[0].properties.EE2024 + "m").addTo(map);
                });

            // Change the cursor to a pointer when the mouse is over the stations layer.
            map.on('mouseenter', 'TubeEntryExit2', function () {
                  map.getCanvas().style.cursor = 'pointer';
            });

            // Change it back to a pointer when it leaves and remove the popup window.
            map.on('mouseleave', 'TubeEntryExit2', function () {
                map.getCanvas().style.cursor = '';
                mypopup.remove();
            });


    });


</script>

</body>
</html>
```




The final map should appear as above, with the interactive time slider that changes the data year that is displayed and updates the size of the tube station circles and the labels. There should also be a popup table that appears when you move your pointer over different tube stations (note the latest version of the annual entry and exit data goes up to 2024).

## Choropleth Mapping Example: House Prices Per Square Metre

Mapbox can visualise lots of different types of spatial data. The Tube map example uses point data, which we mapped using proportional circles, and line data for the basemap. Another common example spatial data type is zonal/polygon data, such as demographic, built environment and other socio-economic data. An example of mapping this kind of data is shown in "Mapbox_example5_HousePriceMap.html".




We will look in more detail at this example in a later practical in a couple of weeks. For now, we will discuss some of the principle techniques used here. You will see that the zonal data has been clipped to urban boundaries. This is a useful technique for urban data visualisation, so that the user can see how the variable being visualised changes between cities, towns and rural areas. The urban outlines data comes from Ordnance Survey data on Digimap (Meridian 2 dataset). The OS also has a more detailed "OS Open Built-up Areas" dataset. The urban outlines are then used with a clip operation in GIS software.

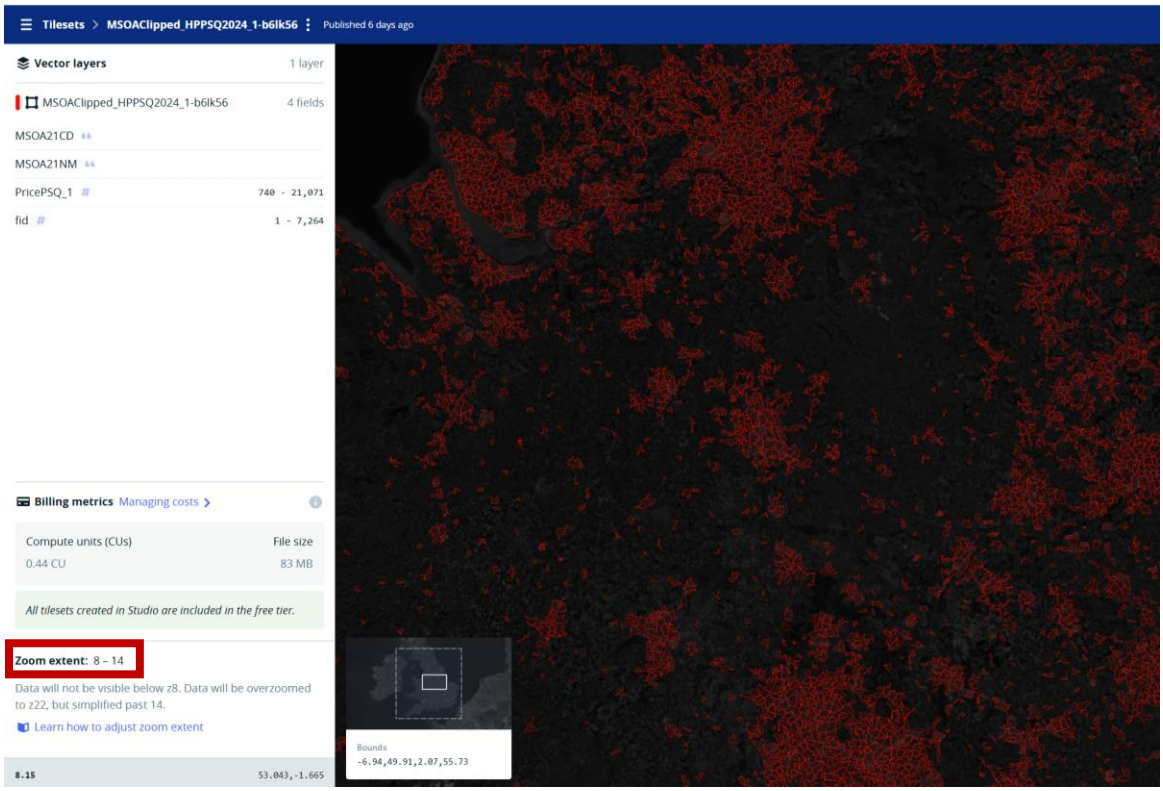
The House Price per Square Metre data comes from combining the Land Registry Price Paid data with the Energy Performance Certificate floorspace data, achieved in a research project at CASA UCL- https://data.london.gov.uk/dataset/house-price-per-square-metre-in-england-and-wales-epo9w/

The mapping of the data is done in Mapbox Studio, then the interactivity has been achieved by adding a transparent layer on top of the basemap style which is used for user interactions. Look at the code for "Mapbox_example5_HousePriceMap.html" to see more details.

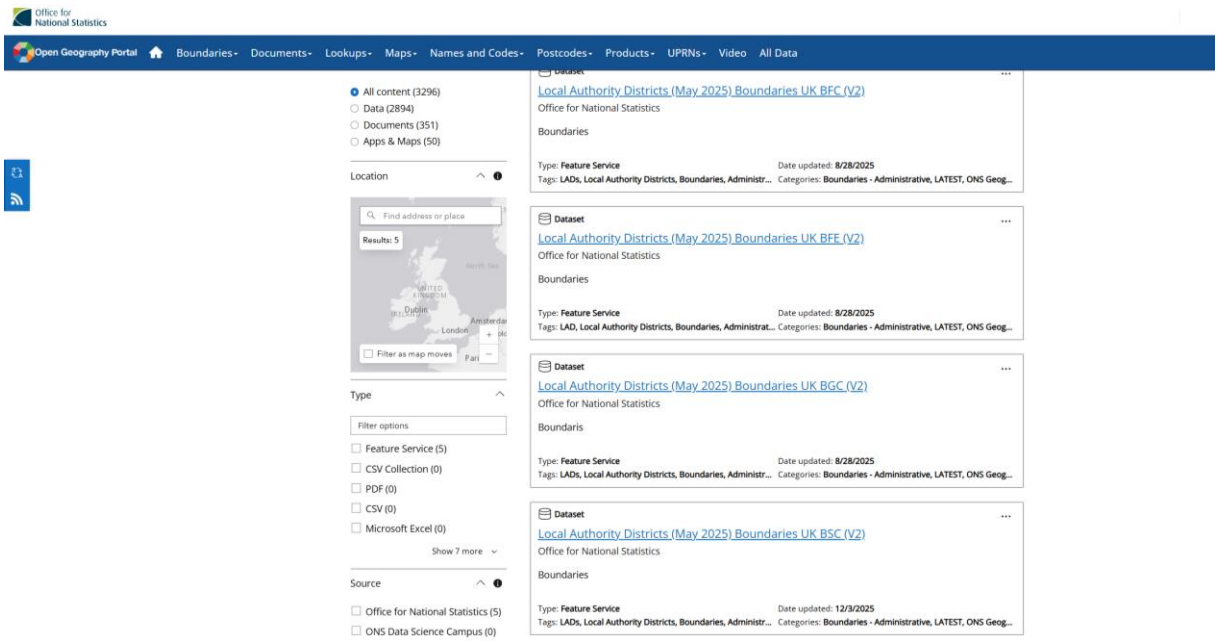
## Zoom Levels and Level of Detail for Vector Tiles

A challenge when mapping more complicated geometry using vector tiles is that there is a maximum number of features that can be included in each vector tile. This means it is more difficult to visualise detailed vector data as you zoom out, and Mapbox can limit the visibility of layers at low zoom levels. The MSOA data used in the House Price Map has a minimum zoom level of 8, which means

that the layer is not visible at zoom levels below this-

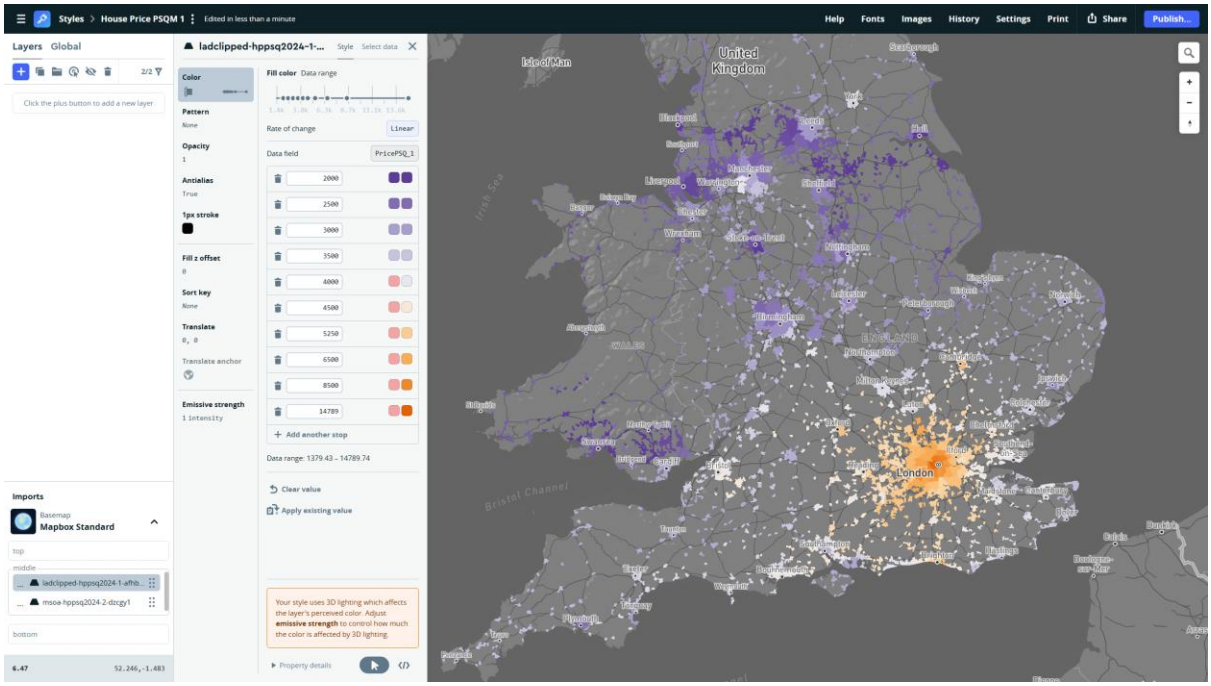



Usually the best way to solve this issue is to create an additional layer that has been simplified and has lower resolution geometry. In this example there are two house price layers: an MSOA layer when you zoom in, and a much lower detail Local Authority layer when you zoom out. The Local Authority layer uses the lowest detail zones from the ONS Geoportal. The boundary layers on the ONS Geoportal are marked BFC (full level of detail, clipped to coastline); BGC (generalised, clipped to coastline) and BSC (super-generalised, clipped to coastline). Here we have used the super-generalised option to get the most simplified vector data. This is then visible on Mapbox across all zoom levels.



Note that another method to control zoom levels on vector iles to use the Mapbox software Tippecanoe to control how features are dropped at low zoom levels, allowing a wider zoom extent.

When we view the House Price data in Mapbox Studio, you can see there are two layers, the Local Authority layer and the MSOA layer. The Local Authority layer has a max zoom of 8, and the MSOA layer has a min zoom of 8, so the two layers 'swap over' as the user zooms in and the data is visible at all zoom levels-




The data layers have also been placed in the 'middle' section (bottom-left) of the layers panel. This means that Mapbox place labels will appear above the layer, and so the user is provided with a hierarchy of place labels without us having to do much work.

There are quite a few techniques being used in this example which we will revisit in the advanced mapping session in two week's time. It is useful to introduce these today so you get an idea of the kind of mapping that Mapbox can achieve.

## Further Materials on Mapbox

We will look at further Mapbox examples next week, including some 2.5D mapping and camera controls. Mapbox have created a powerful and flexible set of tools that can be used on lots of different types of spatial data, and offer users a variety of different interaction possibilities.

A good overview of the entire Mapbox tool suite is found here-
https://docs.mapbox.com/help/getting-started/

There are also some great Mapbox.gl JS examples online that you can look at-
https://docs.mapbox.com/mapbox-gl-js/example/

These examples include some nice feature like updating a layer dynamically by zoom level-
https://www.mapbox.com/mapbox-gl-js/example/updating-choropleth/

A time-slider of event data-
https://www.mapbox.com/mapbox-gl-js/example/timeline-animation/

A slideshow of camera ‘fly-to’ events-
https://www.mapbox.com/mapbox-gl-js/example/scroll-fly-to/

The development of Mapbox tools can also be tracked via Github. This is a good way to understand what new features are coming in future versions, and common bugs developers have identified-
https://github.com/mapbox/mapbox-gl-js/issues