FIT3179 Data Visualisation

Week 8 Studio: Create Maps with Vega-Lite

1. Spatial Data Formats

GeoJSON, TopoJSON and shapefile are commonly used file formats for spatial data that are essential to understand.

1.1 GeoJSON

One of the most popular spatial data formats is GeoJSON. GeoJSON is a JSON text file containing specialised attributes for geospatial data storage. The GeoJSON standard format is specified under IETF's (Internet Engineering Task Force) RFC 7946. The GeoJSON format consists of several geometry objects (figure 1).

- Point: a single geospatial position.
- LineString: is an array of two or more positions.
- Polygon: an array of linear ring coordinate arrays.
- MultiPoint: an array of positions.
- MultiLineString: an array of LineString coordinate arrays.
- MultiPolygon: an array of Polygon coordinate arrays.
- GeometryCollections: a collection of geometry objects.

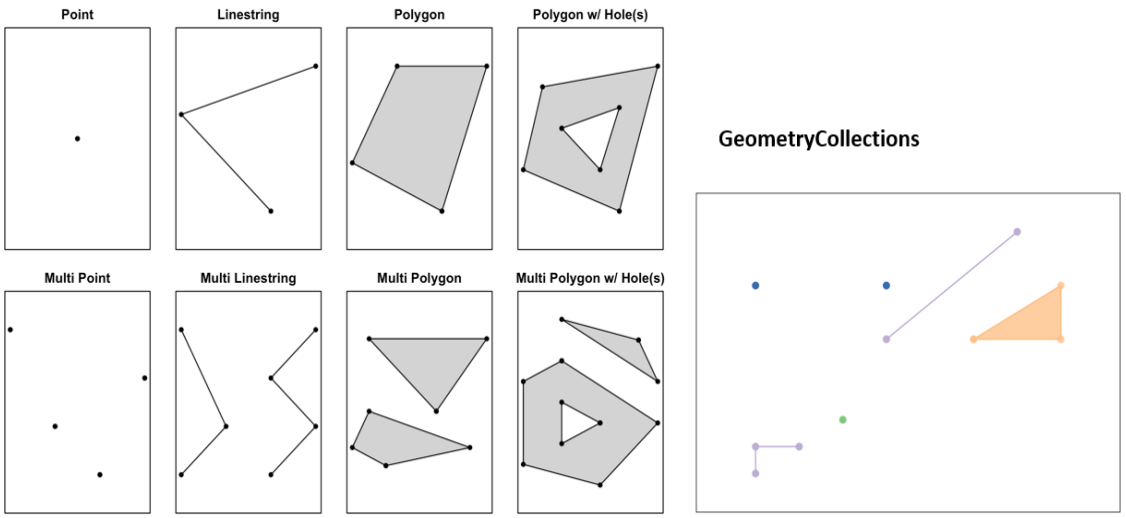


Figure 1. Geometry objects. Source.

1.2 TopoJSON

TopoJSON is an extension of GeoJSON that encodes topology. Rather than representing geometries discreetly, geometries in TopoJSON files are stitched together from shared line segments called arcs.

The main benefit of using the TopoJSON file is that it improves shape simplification by avoiding artefacts that would be caused by simplifying shapes independently. It also enables applications like map colouring and selective meshing and makes the format more compact by removing redundant coordinates.

TopoJSON is the only spatial file format that is directly supported in Vega-Lite v5.

The image below shows an example of a TopoJSON file (with the JSON viewer plugin). You may download the file here or view the spatial data on a map here. From this figure we can see that the TopoJSON consists of a single topology object, declared on line 5. The "arcs" property defined between line 6 and line 34379 contains an array of arrays of positions; all arcs have an index, which can be referred to later in the geometries. The "transform" property is optional and normally consists of a scale and translation applied to all geometries. The "objects" property defines all geometry objects including "Point", "LineString", "Polygon", "GeometryCollection", etc. In this example, there is only one geometry object with a type "Geometry Collection", which represents a collection of geometry objects. The object name is "ne_110m_admin_0_countries". This name will be used frequently in our Vega-Lite code as a value of the "feature" property. (We will come back to this when we explain the Vega-Lite code in section 3.2.) Inside this collection, each geometry object has (1) an "arcs" property that consists of indices referring to arcs defined above; (2) a type of this geometry object; and (3) properties of this geometry object such as the type "country" and name "USA". (This name will be used as a key to join with other data sources, more information can be found in section 4.2.)

```
// 20220910154830
// https://raw.githubusercontent.com/JiazhouLiu/FIT3179/main/VegaLite/3_choropleth_map/js/ne_110m_admin_0_countries.topojson

{
  "type": "Topology",
  "arcs": [↔],
  "transform": {↔},
  "objects": {
    "ne_110m_admin_0_countries": {
      "type": "GeometryCollection",
      "geometries": [
        {
          "arcs": [↔],
          "type": "MultiPolygon",
          "properties": {↔}
        },
        {
          "arcs": [↔],
          "type": "Polygon",
          "properties": {↔}
        },
        {↔},
        {
          "arcs": [↔],
          "type": "MultiPolygon",
          "properties": {↔}
        },
        {
          "arcs": [↔],
          "type": "MultiPolygon",
          "properties": {
            "featurecla": "Admin-0 country",
            "scalerank": 1,
            "LABELRANK": 2,
            "SOVEREIGNT": "United States of America",
            "SOV_A3": "US1",
            "ADM0_DIF": 1,
            "LEVEL": 2,
            "TYPE": "Country",
            "ADMIN": "United States of America",
            "ADM0_A3": "USA",
            "GEOU_DIF": 0,
            "GEOUNIT": "United States of America",
            "GU_A3": "USA",
            "SU_DIF": 0,
            "SUBUNIT": "United States",
            "SU_A3": "USA",
            "BRK_DIFF": 0,
            "NAME": "United States of America",
```


1.3 Shapefile

Shapefile is another spatial data format. This data format is owned by Esri, the leading company in Geographic Information System (GIS) software. ArcGIS (a widely used GIS) is one of their products. A shapefile consists of a group of files stored in one directory. Shapefile contains at least .shp, .shx, and .dbf files.

- .shp: contains geometry objects.
- .dbf: contains attributes in dBase format.
- .shx: contains indices linking the geometry objects with their attributes.

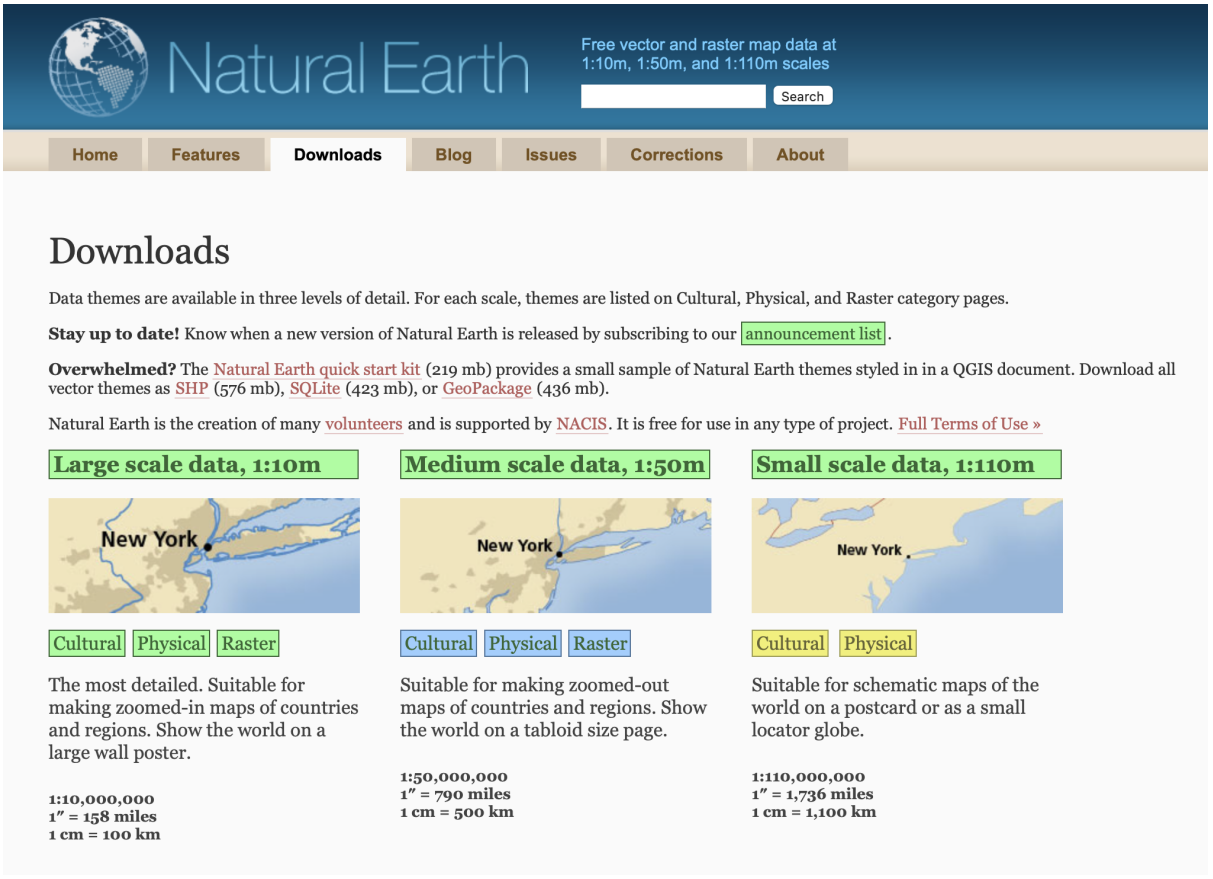
The shapefile format also often has other files, such as .prj, .cpg, or .qpg. Always keep all files of a shapefile dataset in the same directory.

2. Obtaining Spatial Data

We first download multiple layers of spatial data, and then convert the layers to the TopoJSON format, which is required for Vega-Lite.

2.1 Downloading Natural Earth Data

https://www.naturalearthdata.com is an excellent resource of spatial data for maps of large areas, such as countries, continents or the entire world. Natural Earth data is distributed in the shapefile format, and available at three different scales with varying information density. 1:10 million data is best for zoomed-in maps of countries and regions; 1:50 million data is best for zoomed-out maps of countries and regions or for large world maps; 1:110 million is best for world maps.



Go to https://www.naturalearthdata.com/downloads/, then under "Small scale data, 1:110m" > select "Cultural" > and download "Admin 0 – Countries". This will download the file "ne_110m_admin_0_countries.zip". This zip file contains one shapefile with country borders. You do not need to unzip the file.

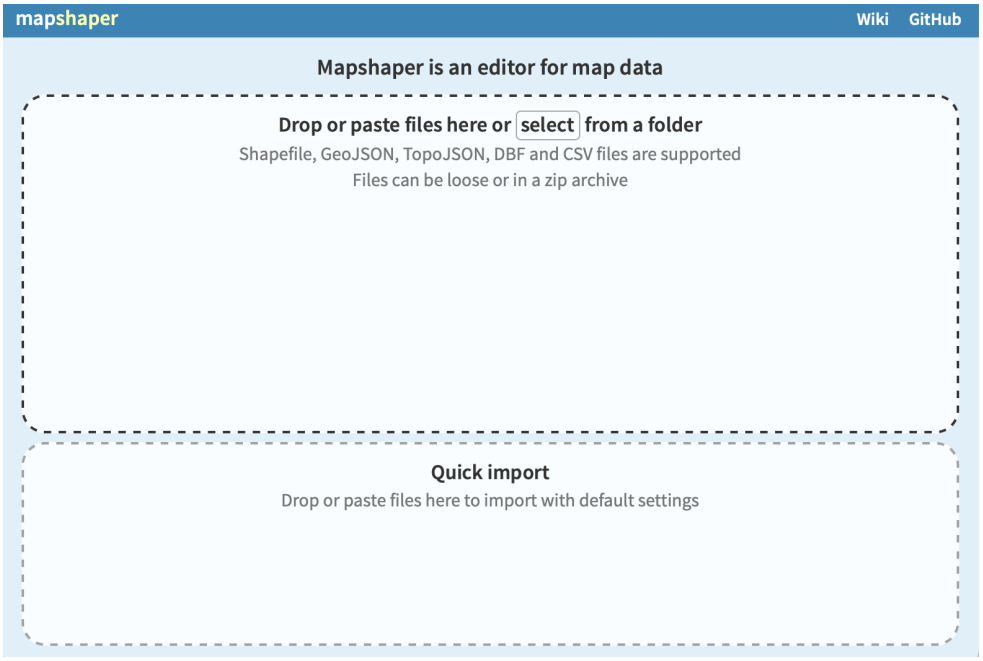
Also download a graticule (that is, lines of constant longitude and latitude): Scroll to the bottom of https://www.naturalearthdata.com/downloads/110m-physical-vectors/ and select “Download 30” for a graticule with a spacing of 30 degrees between meridians and parallels.

Note: There are many useful online websites providing various types of spatial data. To find spatial data with a search engine, include the spatial region (e.g., global, Australia, America) and the file type (e.g., shapefile, GeoJSON, TopoJSON) in your query. Here are useful websites that provide spatial data:

- Global scale: https://www.naturalearthdata.com
- Australia Spatial Data: https://www.abs.gov.au/ausstats/abs@.nsf/mf/1270.0.55.001
- Victoria Suburbs: https://data.gov.au/dataset/ds-dga-af33dd8c-0534-4e18-9245-fc64440f742e/details

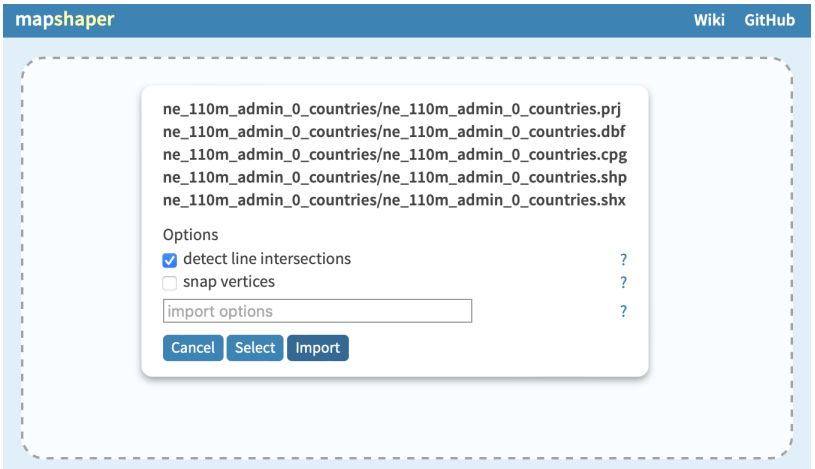
2.2 Converting spatial data to TopoJSON

Mapshaper is a simple and sophisticated tool for converting spatial data in different formats to the TopoJSON format: https://mapshaper.org/. You can use other tools, e.g., QGIS, geopandas package in python, etc., but mapshaper is probably the easiest to use.

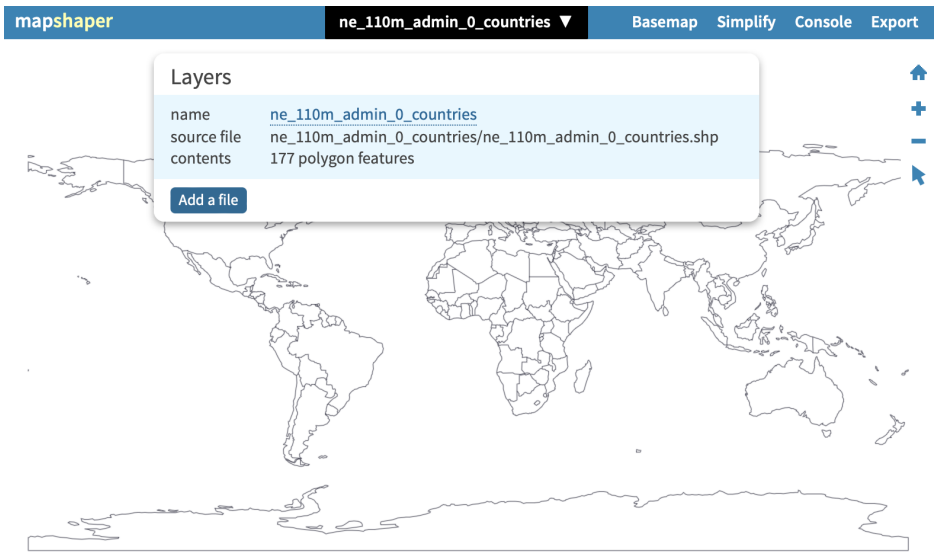


Important: Your web browser likely will have “unzipped” the downloaded ZIP files. With mapshaper, it is important to open a ZIP file with all files that make up a shapefile. If you only open the .shp file, all data attributes will be lost. Therefore retrieve the original downloaded ZIP file, or create a new ZIP file with all files that make up the shapefile.

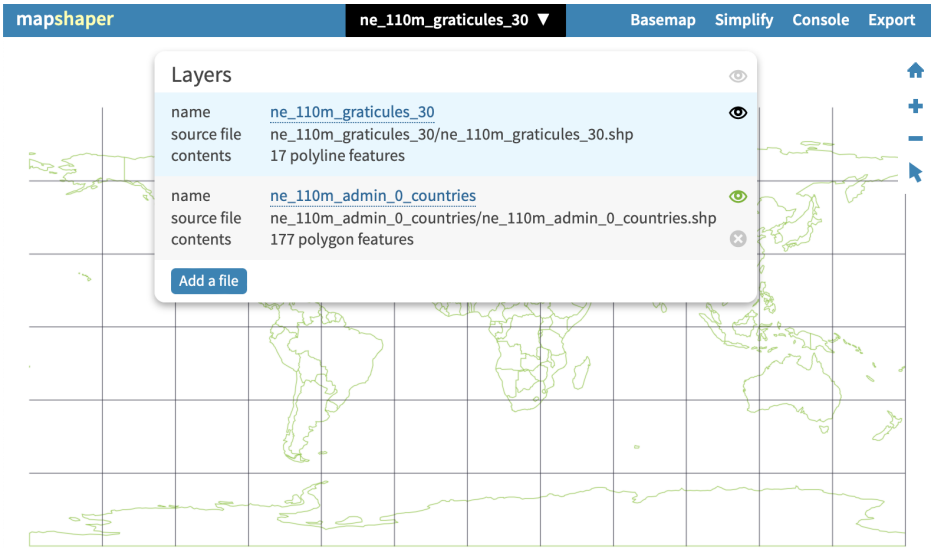
Go to https://mapshaper.org/, then click “Select” and choose the “ne_110m_admin_0_countries.zip” file that you downloaded. There will be a panel popping up. Click “Import” to continue.



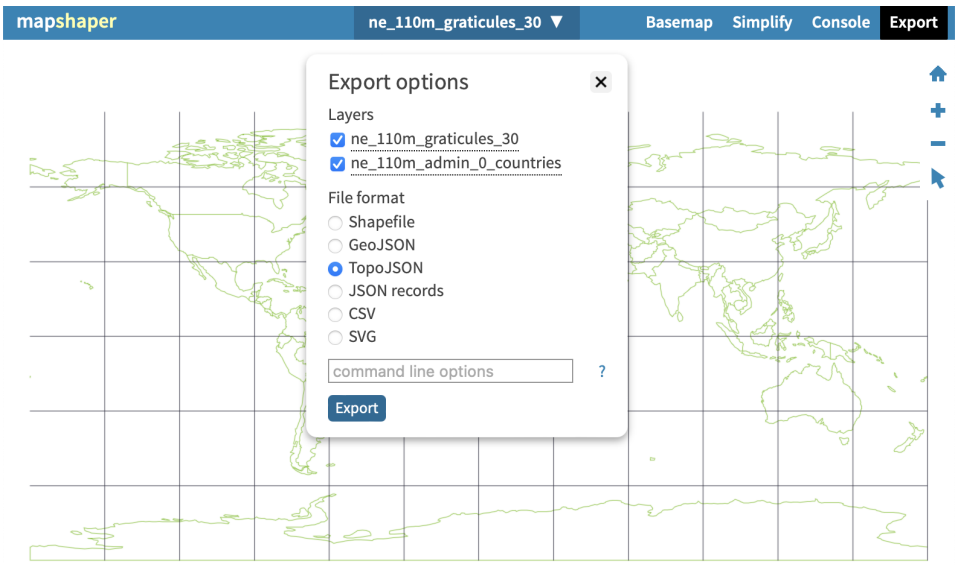
Import the graticule lines: Click on the layer name and then click on “Add a file” and select the “ne_110m_graticules_30.zip” file.



To see both layers, click on the layer name again and toggle the eye icon on. Note that the order of layers can be changed by dragging them.



Click “Export” on the top-right corner, and select “TopoJSON” > “Export” to create a TopoJSON file. Note that the layers to export can be selected.



2.3 Clipping a Spatial Geometry to Match Your Need

Polygon clip and erase are essential map making operations. For example, if you want to create a map of Australia, but have a spatial dataset of the entire world, clipping reduces the size of your TopoJSON file, and your Vega-Lite map may load considerably faster. Mapshaper (https://mapshaper.org/) provides a clipping command in the console, as well as a direct crop selection on the webpage. Here, we show an example of how to crop a section from a world map to create a map of Australia.

Command Line Tool

In the top-right corner of mapshaper.org click on the "Console" button next to the "Export" button to enable the mapshaper command line tool (details can be found in the mapshaper reference). We use the clip command to crop our data. Type `-help clip` to see details about its usage. We will use the "bbox" option for the cropping later.

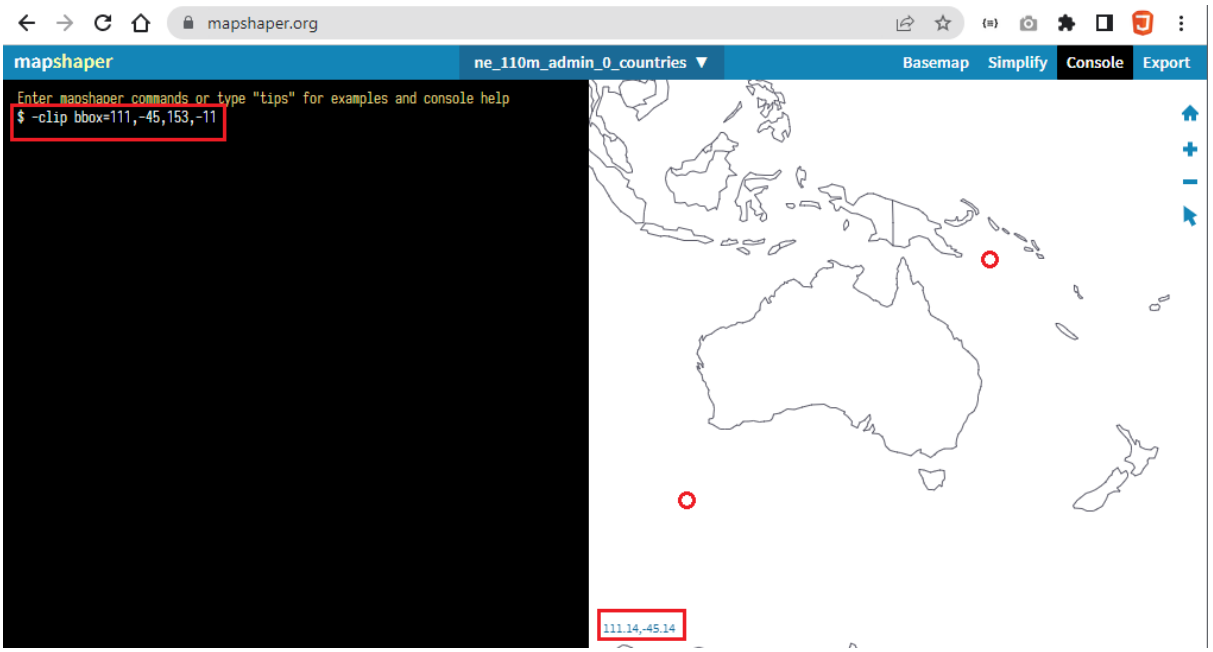
```
$ -help clip
COMMAND
-clip           use a polygon layer to clip another layer

OPTIONS
<source>        shortcut for source=
source=         file or layer containing clip polygons
remove-slivers  remove sliver polygons created by clipping
bbox=           comma-sep. bounding box: xmin,ymin,xmax,ymax
bbox2=          experimental fast bbox clipping
name=           rename the edited layer(s)
target=         layer(s) to target (comma-sep. list)
+, no-replace   retain both input and output layer(s)

EXAMPLE
  $ mapshaper states.shp -clip land_area.shp -o clipped.shp
```

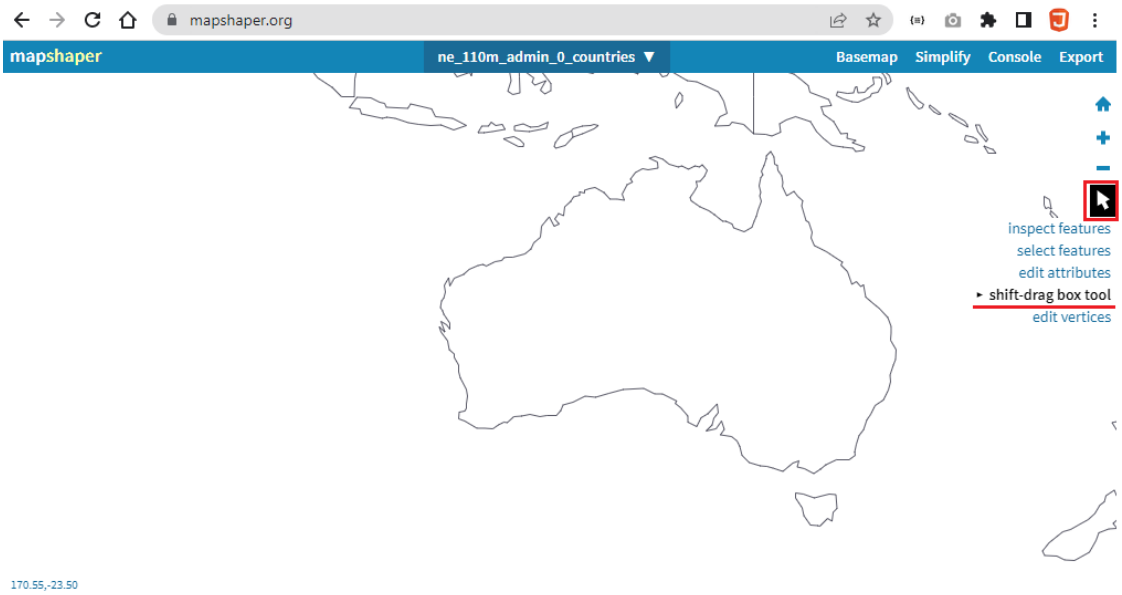
According to the bbox description, we need xmin, ymin, xmax, and ymax as arguments. To find values for Australia on the map, hover over a point on the map to see the x/y coordinates of your cursor (you can zoom or pan for a better view). We can find xmin (111) and ymin (-45) at the bottom-left corner of Australia and xmax (153) and ymax (-11) at the top-right corner. So we type "`-clip bbox=111,-45,153,-11`" in the command line. After confirming with the enter key, the world map will be cropped to a map of Australia and its surroundings.

Note: NaturalEarth data is in spherical longitude and latitude coordinates. So the x/y coordinates are longitude and latitude values.

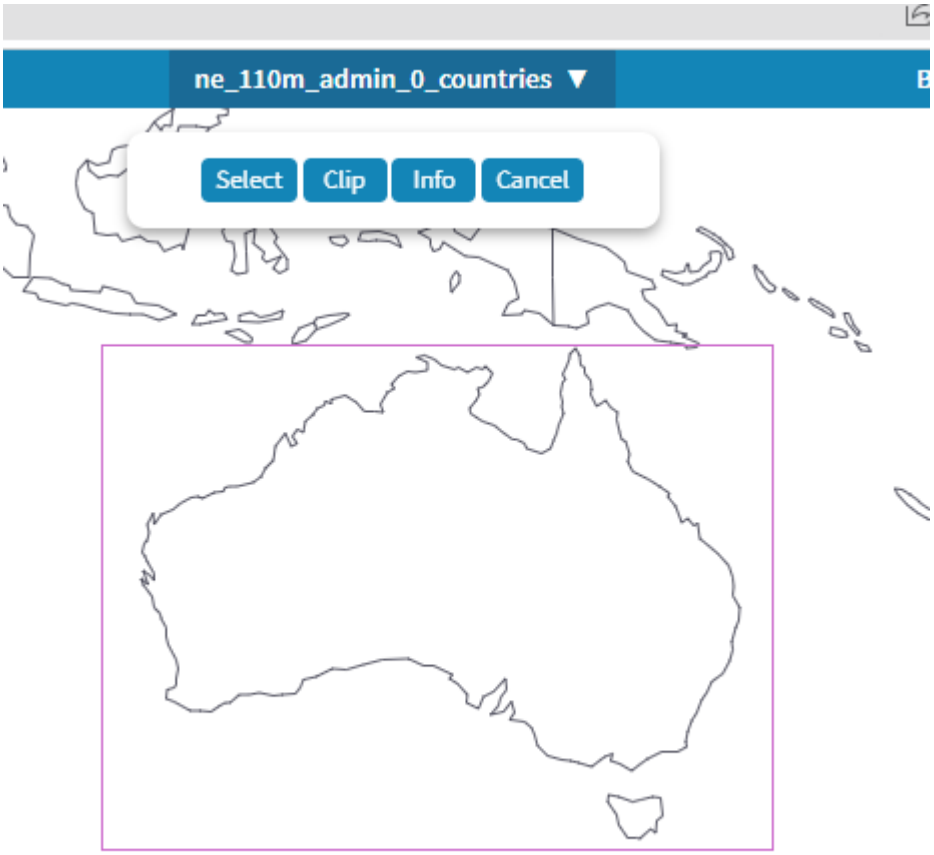


Web UI selection

Alternatively, you can also click on the  icon and select the “shift-drag box tool”.



Hold the shift button and drag an area that contains Australia. Then select “Clip” on the pop-up menu at the top. Now the world map is cropped to a map of Australia and its surroundings.


ne_110m_admin_0_countries ▼
Select
Clip
Info
Cancel

3. Proportional Symbol Maps with Vega-Lite

In this section, we build a simple proportional symbol map in Vega-Lite using an earthquake dataset. The dataset contains earthquakes of magnitude 4 or larger between 4 Oct 2020 and 10 Oct 2. The data is downloaded from https://earthquake.usgs.gov/earthquakes/search/. The downloaded data is available as a CSV table here: https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/2_symbol_map/data/earthquake.csv

The main attributes that we use include:

- latitude
- longitude
- mag
- depth

We will also use the TopoJSON data (from Section 2.2) to create a base map for our proportional symbol map. The TopoJSON data is available here: https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/2_symbol_map/js/ne_110m_admin_0_countries.topojson

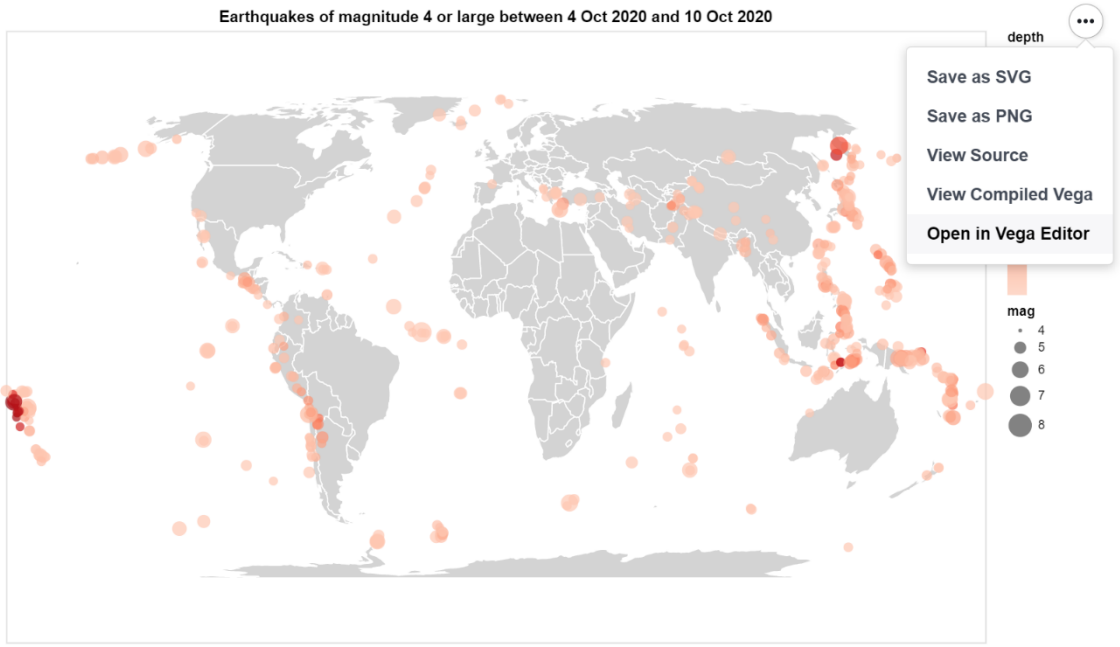
3.1 Overview

The final proportional symbol map will look like this: https://fit3179.github.io/Vega-Lite/2_symbol_map/

The example GitHub repository is available here: https://github.com/FIT3179/Vega-Lite/tree/main/2_symbol_map

3.2 Understand the Vega-Lite JSON file

You can see the final map here https://fit3179.github.io/Vega-Lite/2_symbol_map/. From the visualisation page, click “Open in Vega Editor” (as shown below) to edit the JSON code.



Here is the JSON specification. We explain it line by line below.

```
1  {
2    "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
3    "width": 800,
4    "height": 450,
5    "title": "Earthquakes of magnitude 4 or large between 4 Oct
     2020 and 10 Oct 2020",
6    "projection": {"type": "equalEarth"},
7    "layer": [
8      {
9        "data": {
10         "url":
     "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/2_symbo
     l_map/js/ne_110m_admin_0_countries.topojson",
11         "format": {"type": "topojson", "feature":
     "ne_110m_admin_0_countries"}
12       },
13       "mark": {"type": "geoshape", "fill": "lightgray", "stroke":
     "white"}
14     },
```

```
15      {
16        "data": {
17          "url":
    "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/2_symbo
    l_map/data/earthquake.csv"
18        },
19        "mark": {"type": "circle", "tooltip": {"content": "data"}},
20        "encoding": {
21          "longitude": {"field": "longitude", "type":
    "quantitative"},
22          "latitude": {"field": "latitude", "type":
23  "quantitative"},
24          "size": {
25            "field": "mag",
26            "type": "quantitative",
27            "title": "Magnitude (Richter scale)",
28            "scale": {"domain": [4, 8]}
29          },
30          "color": {
31            "field": "depth",
32            "type": "quantitative",
33            "title": "Depth in km",
34            "scale": {"scheme": "reds"}
35          }
36        }
37      }
38    ]
    }
```

Lines 2-5: We specify the Vega-Lite version, the width and height of the map, as well as the title of the visualisation.

```
2   "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
3   "width": 800,
4   "height": 450,
```

```
5   "title": "Earthquakes of magnitude 4 or large between 4 Oct 2020
     and 10 Oct 2020",
```

Line 6: We set up the map projection as “equalEarth”, which is an area-preserving map projection for world maps. Vega-Lite supports a few different cartographic projections, including the equirectangular projection, the orthographic projection, etc. See this page for all supported projections: https://vega.github.io/vega-lite/docs/projection.html. Change the projection type (in Vega Editor) and observe the differences.

```
6   "projection": {"type": "equalEarth"},
```

Lines 7–37 define the two layers of the map. The first layer consists of the outline of countries (defined in lines 8–14), and the second layer consists of proportional symbols (defined in lines 15–36). Let’s have a look at the map layer first. Copy and paste the code below to add the map layer.

```
1   {
2     "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
3     "width": 800,
4     "height": 450,
5     "title": "Earthquakes of magnitude 4 or large between 4 Oct 2020 and 10
    Oct 2020",
6     "projection": {"type": "equalEarth"},
7     "layer": [
8       {
9         "data": {
10          "url": "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/2
    _symbol_map/js/ne_110m_admin_0_countries.topojson",
11          "format": {"type": "topojson", "feature": "ne_110m_admin_0_countri
    es"}
12        },
13        "mark": {"type": "geoshape", "fill": "lightgray", "stroke": "white"}
      }]
14  }
```

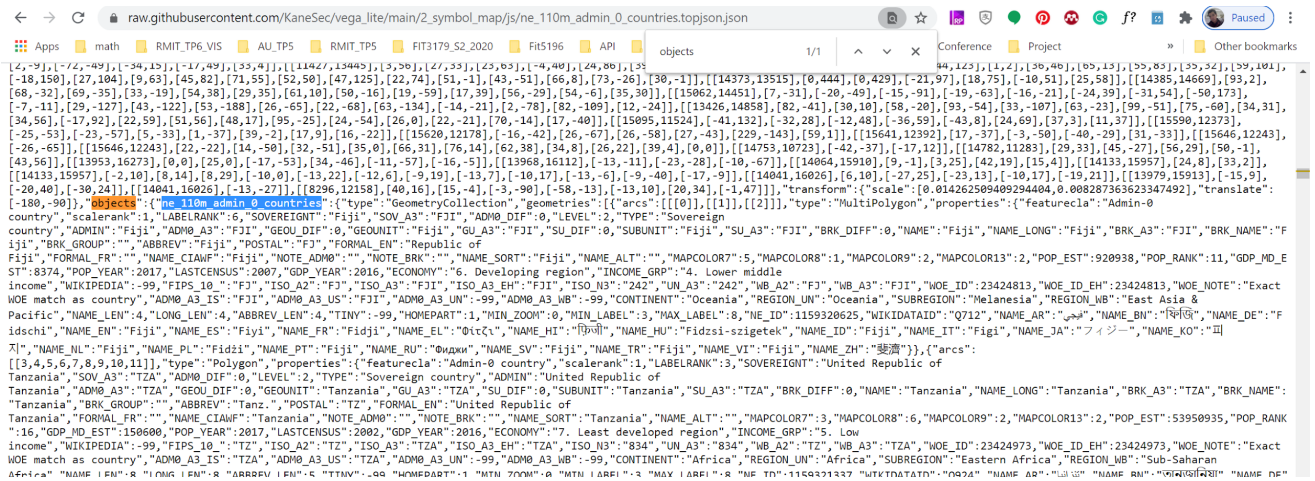
15	

This first layer defines our base map:

- url: this is the TopoJSON file (that we included in our GitHub repository) – please check the GitHub repository in Section 3.1 for the project layout.
- format > type: topojson
- format > feature: "ne_110m_admin_0_countries": this is the name of the TopoJSON geometry object stored in a TopoJSON file as explained in section 1.2. In general, you need to understand your TopoJSON file to find such information.

Tips

1. to quickly identify this name in your TopoJSON file, just search “objects” in the TopoJSON file, the key under that could be used as your feature name).
2. Copy the TopoJSON content to an online JSON formatter (e.g., jsonformatter.curiousconcept.com). Then check the formatted JSON file to understand the structure of the TopoJSON file.



- Mark: this defines the encoding of our map:
  - "type": "geoshape",
  - "fill": "lightgray" (this defines the fill colour of each country)
  - "stroke": "white" (this is the outline colour of each country)

Lines 15–36 define our second layer – earthquake data presented as proportional symbols.

```
15   {
16         "data": {
17           "url":
     "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/2_symbo
     l_map/data/earthquake.csv"
18         },
19         "mark": {"type": "circle", "tooltip": {"content": "data"}},
20         "encoding": {
21           "longitude": {"field": "longitude", "type":
     "quantitative"},
22           "latitude": {"field": "latitude", "type":
23   "quantitative"},
24           "size": {
25             "field": "mag",
26             "type": "quantitative",
27             "title": "Magnitude (Richter scale)",
28             "scale": {"domain": [4, 8]}
29           },
30           "color": {
31             "field": "depth",
32             "type": "quantitative",
33             "title": "Depth in km",
34             "scale": {"scheme": "reds"}
35           }
36         }
       }
```

- "data": the earthquake data (CSV file). Vega-Lite also supports values separated by tabs and custom delimiters, see here: https://vega.github.io/vega-lite/docs/data.html

- “mark”: this defines the mark as “circle”, and adds tooltips that show data details. For more about custom-built tooltips, see https://vega.github.io/vega-lite/docs/tooltip.html.
- Encodings:
    - “longitude”: this is the longitude data key required by Vega-Lite to position the circles.
        - field: “longitude”: this is the column name in our CSV data
    - “latitude”: similar to “longitude”.
    - “size”:
        - Field: the attribute name in our CSV file. We select “mag”, which is the magnitude of the earthquake
        - Type: the attribute type is quantitative.
        - Title: a title for the field, here we define the legend for the size channel.
        - Scale – domain: we define the magnitude range to display on the map.
    - “colour”
        - Field: the attribute name in our CSV file. We select “depth”, which is the depth of earthquakes.
        - Type: the attribute type is quantitative.
        - Title: a title for the field, here we define the legend for the colour channel.
        - Scale – scheme: provide a set of named colour palettes for both discrete and continuous colour encodings. For more details: https://vega.github.io/vega/docs/schemes/#reference

Exercise:

Try to modify the encodings and check the corresponding results. For example, use the colour channel to encode the magnitude (“mag” attribute) of the earthquake and use the size channel to encode the duration (“dmin” attribute) of the earthquake.
Also try to add a new encoding: “shape”, and use it for the location source (locationSource attribute). See if there is any warning for this encoding. Do you understand why? Can you fix it?

3.3 The HTML Document and Vega-Lite Map Examples

Check the example GitHub repository (in Section 3.1) to understand the HTML document structure. We defined a div called “symbol_map” in our index.html file.

- Styles.css: we control the size and location of the div.
- Data folder: this includes all data files.
- JS folder: this includes all JavaScript and JSON files. (Some programmers might prefer to put JSON files in the data folder.)

If you have difficulty understanding those files, please check the HTML, CSS and JavaScript tutorials in the previous lab activities.

For more examples of proportional symbol maps or dot maps, see the following:

- One Dot per Airport in the U.S. Overlayed on Geoshape: https://vega.github.io/vega-lite/examples/geo_layer.html
- One Dot per Zipcode in the U.S.: https://vega.github.io/vega-lite/examples/geo_circle.html

4. Choropleth Maps with Vega-Lite

In this section, we build a choropleth map in Vega-Lite using a COVID-19 dataset. The dataset contains the COVID-19 statistical data of different countries on 10 Oct 2020. The data is available here: https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/3_choropleth_map/data/covid_10_10_2020.csv

The main attributes that we use include

- Country
- Active: the number of active cases of each country on 10 Oct 2020
- Confirmed
- Population

We will also use the TopoJSON data (from Section 2.2) to create a global map. The TopoJSON data is available here: https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/3_choropleth_map/js/ne_110m_admin_0_countries.topojson

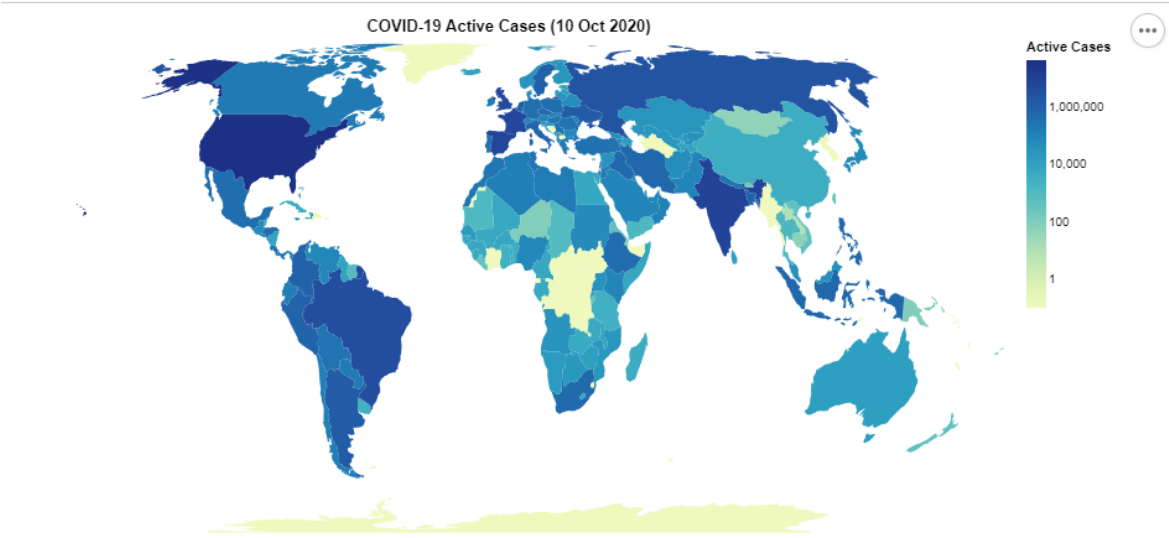
4.1 Overview

The final Choropleth map will look like this: https://fit3179.github.io/Vega-Lite/3_choropleth_map/

The example GitHub repository is available here: https://github.com/FIT3179/Vega-Lite/tree/main/3_choropleth_map

4.2 Understand the Vega-Lite JSON file

Here, we walk through the process of creating this map. On the visualisation page at https://fit3179.github.io/Vega-Lite/3_choropleth_map/, click "Open in Vega Editor" (as shown below).



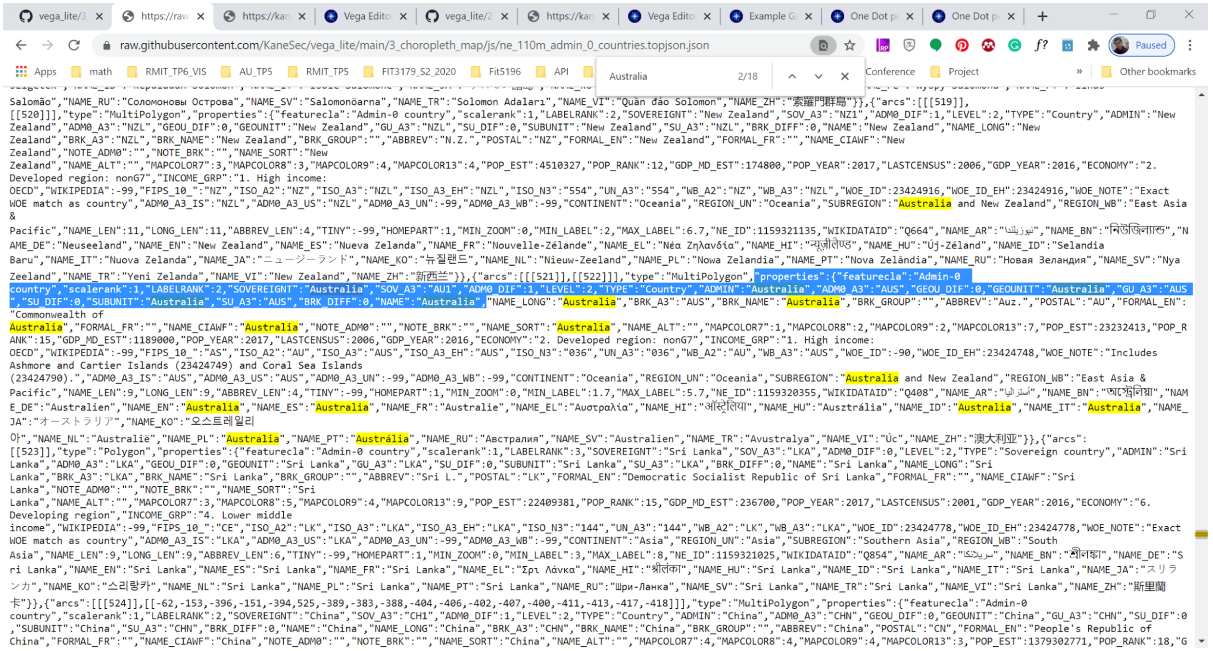

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
  "title": "COVID-19 Active Cases (10 Oct 2020)",
  "width": 800,
  "height": 400,
  "projection": {"type": "equalEarth"},
  "data": {
    "url": "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/3_choropleth_map/js/ne_110m_admin_0_countries.topojson",
    "format": {"type": "topojson", "feature": "ne_110m_admin_0_countries"}
  },
  "transform": [
    {
      "lookup": "properties.NAME",
      "from": {
        "data": {
          "url": "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/3_choropleth_map/data/covid_10_10_2020.csv"
        },
        "key": "Country",
        "fields": ["Active"]
      }
```

```
20      },
21      {"calculate": "datum.Active + 0.1", "as": "Active Cases"}
22    ],
23    "mark": {"type": "geoshape"},
24    "encoding": {
25      "color": {
26        "field": "Active Cases",
27        "type": "quantitative",
28        "scale": {"type": "log"}
29      },
30      "tooltip": [
31        {"field": "properties.NAME", "type": "nominal", "title":
32  "Country"},
          {"field": "Active", "type": "quantitative"}
33      ]
34    }
35  }
36
```

Lines 2–10: this is similar to our proportional symbol map (where we defined a base map layer).

Lines 11–23: This is where we define our data to map. We will map the “country” column from our CSV file to the “country name” defined in the TopoJSON file.

- "lookup": "properties.NAME": this is the key where we stored the country name in our TopoJSON file. To find the name of this data key in your own TopoJSON file, you can search for a data example in your TopoJSON file (e.g., if you are plotting data based on countries, search for a country name – “Australia”; if you are plotting suburbs in Victoria, search for a suburb name – “Caulfield”). In our example, the country name is stored in the dictionary – “properties” > with the key “NAME”.



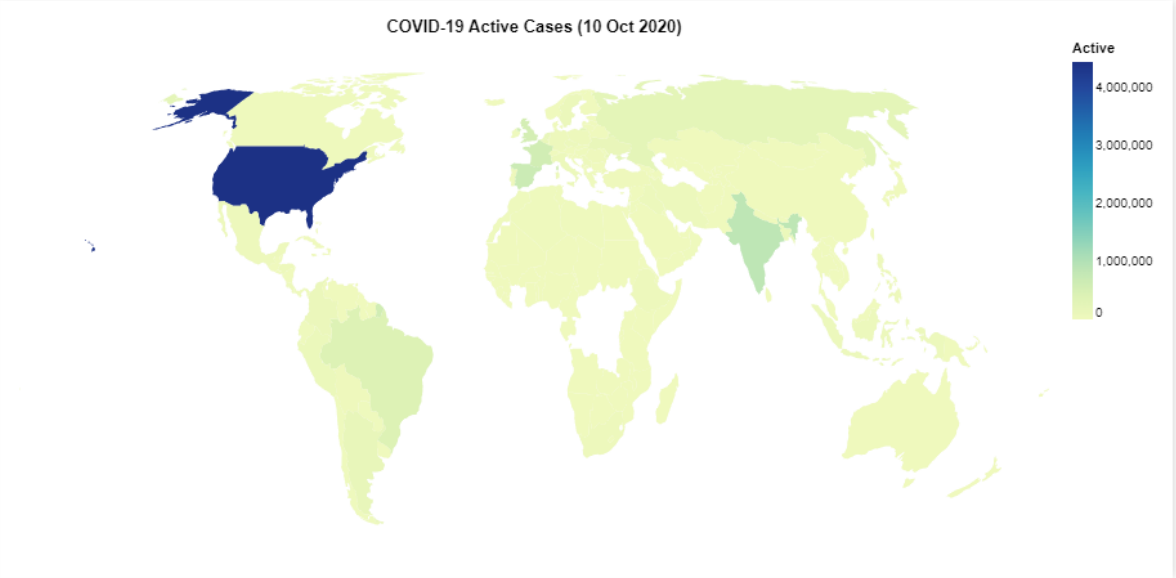
- "from": this is related to the CSV data to be plotted
    - "data": "url": The location of our CSV file.
    - "key": "Country": This is the column name in our CSV file that is going to map the country name in our TopoJSON file.
    - "fields": ["Active"]: This contains all the column names (from the CSV file) that we use in our visualisation mappings. Here we only included the active cases.

Log and Linear Scales

While the remaining lines are easy to understand, please note that we defined a log scale for colour mapping. The codes and visualisation without the log scale are shown below. Please check the visualisation and think about it – why is the log scale better than a linear scale in this case? Also, why do we include "`datum.Active + 0.1`" in line 22 of the original code? Remove the "`+ 0.1`" term and check the visualisation output.

```
1  {
2    "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
3    "title": "COVID-19 Active Cases (10 Oct 2020)",
4    "width": 800,
```

```
5     "height": 400,
6     "projection": {"type": "equalEarth"},
7     "data": {
8       "url": "https://raw.githubusercontent.com/FIT3179/Vega-Lite
      /main/3_choropleth_map/js/ne_110m_admin_0_countries.topojson",
      "format": {"type": "topojson", "feature": "ne_110m_admin_0_
9     countries"}
      },
10    "transform": [
11      {
12        "lookup": "properties.NAME",
13        "from": {
14          "data": {
15            "url": "https://raw.githubusercontent.com/FIT3179/Veg
16    a-Lite/main/3_choropleth_map/data/covid_10_10_2020.csv"
          },
17          "key": "Country",
18          "fields": ["Active"]
19        }
20      }
21    ],
22    "mark": {"type": "geoshape"},
23    "encoding": {
24      "color": {
25        "field": "Active",
26        "type": "quantitative"
27      },
28      "tooltip": [
29        {"field": "properties.NAME", "type": "nominal", "title":
30    "Country"},
        {"field": "Active", "type": "quantitative"}
31      ]
32    }
33  }
34
```



Besides the log scale, Vega-Lite also supports linear, ordinal, band, and point scales. Refer to this page for more details: https://vega.github.io/vega-lite/docs/scale.html

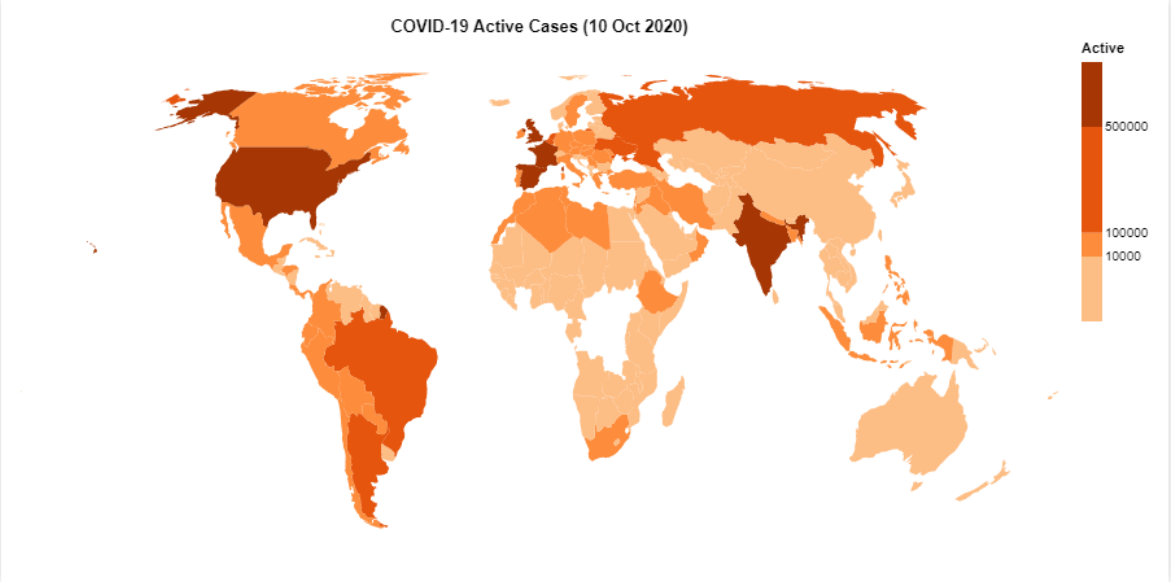
Customised Classification and Colour Scale

We can also customise the classification and the colours. This can be implemented with "scale" in the colour encoding part:

```
1    "scale": {
2          "type": "threshold",
3          "domain": [10000, 100000, 500000],
4          "range": ["#fdbe85", "#fd8d3c", "#e6550d", "#a63603"]
5        }
```

Based on three thresholds defined in the "domain", we divide the number of cases into four ranges [0, 10000], [10000, 100000], [100000, 500000], [500000, infinity]. "range" defines the four colours mapped with these four value ranges.

Please check the visualisation result and the complete Vega-Lite code below:

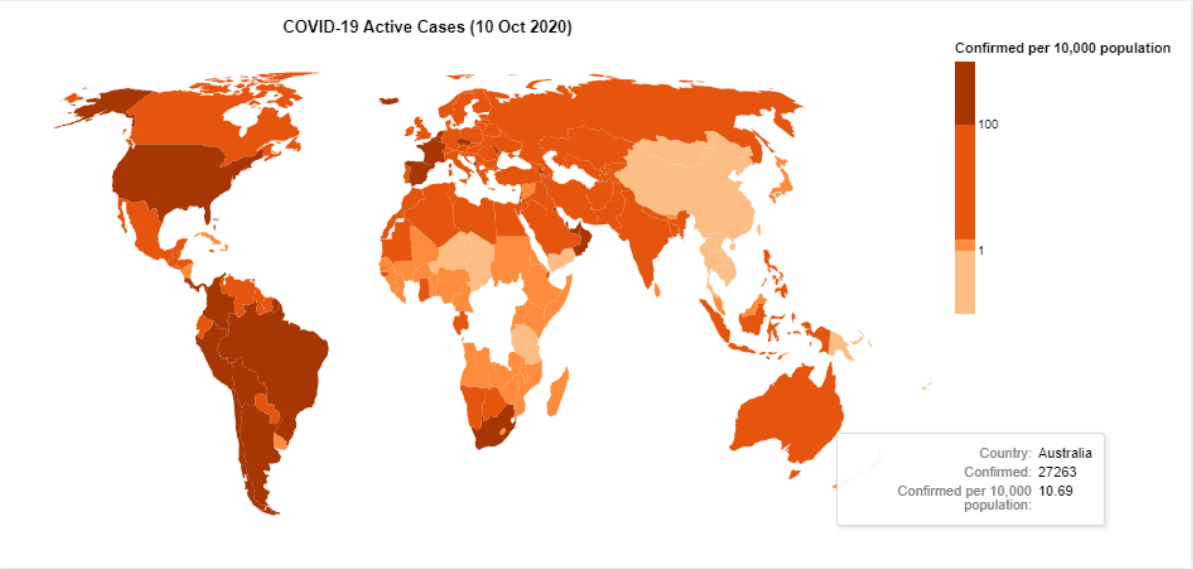



```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
  "title": "COVID-19 Active Cases (10 Oct 2020)",
  "width": 800,
  "height": 400,
  "projection": {"type": "equalEarth"},
  "data": {
    "url": "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/3_choropleth_map/js/ne_110m_admin_0_countries.topojson",
    "format": {"type": "topojson", "feature": "ne_110m_admin_0_countries"}
  },
  "transform": [
    {
      "lookup": "properties.NAME",
      "from": {
        "data": {
          "url": "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/3_choropleth_map/data/covid_10_10_2020.csv"
        },
        "key": "Country",
        "fields": ["Active"]
```

```
      }
    }
  ],
  "mark": {"type": "geoshape"},
  "encoding": {
    "color": {
      "field": "Active",
      "type": "quantitative",
      "scale": {
        "type": "threshold",
        "domain": [10000, 100000, 500000],
        "range": ["#fdbe85", "#fd8d3c", "#e6550d", "#a63603"]
      }
    },
    "tooltip": [
      {"field": "properties.NAME", "type": "nominal", "title":
"Country"},
      {"field": "Active", "type": "quantitative"}
    ]
  }
}
```

Normalise Cases by Population

The choropleth maps created so far are clearly misleading, because they do not normalise data. We need to normalise the numbers based on the population of each country. In Vega-Lite, we can simply do a calculation when loading the data and then plot the choropleth map with this derived data attribute. The following example presents the total confirmed cases per 10,000 population in each country. The colour is mapped based on four ranges [0,1], [1, 10], [10,100] and [100, indefinity]. The result and the code are shown below.




```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
  "title": "COVID-19 Active Cases (10 Oct 2020)",
  "width": 800,
  "height": 400,
  "projection": {"type": "equalEarth"},
  "data": {
    "url": "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/3_choropleth_map/js/ne_110m_admin_0_countries.topojson",
    "format": {"type": "topojson", "feature": "ne_110m_admin_0_countries"}
  },
  "transform": [
    {
      "lookup": "properties.NAME",
      "from": {
        "data": {
          "url": "https://raw.githubusercontent.com/FIT3179/Vega-Lite/main/3_choropleth_map/data/covid_10_10_2020.csv"
        },
        "key": "Country",
        "fields": ["Active", "Confirmed", "Deaths", "Population"]
      }
```

```
20      },
21      {"calculate": "datum.Confirmed/datum.Population * 10000", "as
22  ": "Confirmed per 10,000 population"}
        ],
23      "mark": {"type": "geoshape"},
24      "encoding": {
25        "color": {
26          "field": "Confirmed per 10,000 population",
27          "type": "quantitative",
28          "scale": {
29            "type": "threshold",
30            "domain": [1, 10, 100],
31            "range": ["#fdbe85", "#fd8d3c", "#e6550d", "#a63603"]
32          }
33      },
34      "tooltip": [
35          {"field": "properties.NAME", "type": "nominal", "title": "C
36  ountry"},
            {"field": "Confirmed", "type": "quantitative"},
37          {"field": "Confirmed per 10,000 population", "type": "quant
38  itative", "format": ".2f"}
        ]
39    }
40  }
41
```

Questions

Did you notice that there are some countries (e.g., Congo) shown as blank on the map? Discuss with the tutor and your peers:

- What could be the cause of this?
- What is the best way to handle this?

Neighbouring countries with the same colour are impossible to distinguish. What could be done to fix this?

4.3 The HTML Document and Other Examples

Please check the example GitHub repository (in Section 4.1) to understand the HTML document. We defined a div called “choropleth_map” in our index.html file.

- Styles.css: we control the size and location of the div.
- Data folder: this includes all of our data files.
- JS folder: this includes all of our JavaScript and JSON files. (Some programmers like to put JSON files in the data folder).

For more examples of choropleth maps, please see the following:

- Choropleth of Unemployment Rate per County: https://vega.github.io/vega-lite/examples/geo_choropleth.html
- Other examples: https://vega.github.io/vega-lite/examples/#maps-geographic-displays