Lab 4

Demonstrated to: TBC

Date Demonstrated: TBC

1. Tech Used

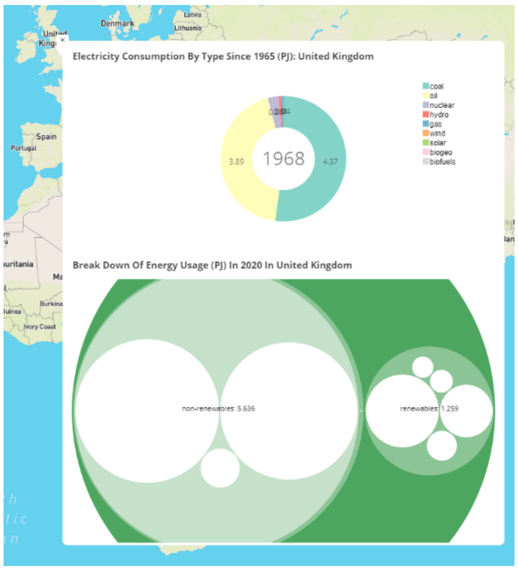
The application is written using a combination of HTML, CSS and JavaScript using version 7 of the D3 library for data visualisation. To render the map, I have used the Mapbox API and used D3 to visualise data on the map; this will be described in more detail in the Map section.

2. Overview

The data I have used in this lab is based on energy data produced by BP as part of their yearly 'Statistical Review of World Energy' (https://www.bp.com/en/global/corporate/energy-economics/statistical-review-of-world-energy.html). The dataset is composed of a large quantity and variety of data regarding energy production and consumption in a large number of countries throughout the world from 1965 up until the current year. The dataset I have used spans from 1965 to 2020 and I have chosen to use data about the energy consumption of each country.

The idea behind my visualisation is that a user can scroll around the map, click on a country and then see how the consumption of energy in that country has changed over the past 55 years. For example, whether a country is consuming energy from more renewable sources compared to previously or to other countries.

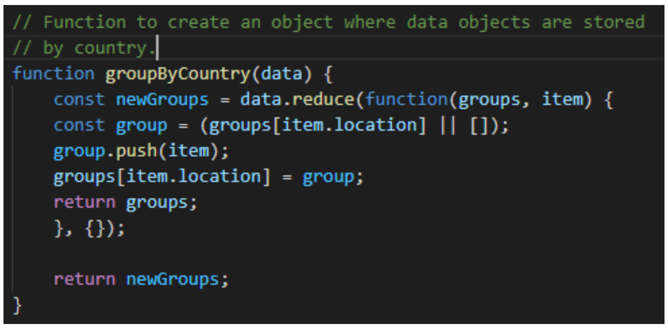
In order to do this, I have used 2 visualisations int the pop up that appears when a user clicks on a country. Firstly, is a pie chart that animates to show the energy consumption break down of each year from 1965-2020; the pie shows one year per second. Secondly is a circle pack chart that shows the split of fuel consumption for 2020 for that country. The carless are grouped into energy type i.e. renewable and non-renewable and placed in a hierarchy whereby the user can zoom in to individual groups to see the energy sources that make up that group.



Pop-up shown to user when they click on a country

3. Data Processing

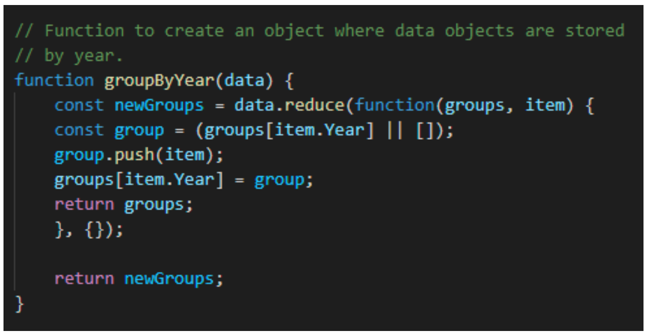
I have used the d3.csv function to read the data in from an external csv file and initially I use a function to group the data by country. To do this I use the same function as I used in lab 3. As the data will be shown by country, I created a function that will group the data given as an argument by country. Using the Array.prototype.reduce() function I loop through each item in the dataset. In the first iteration, an empty array is created and the array element is pushed to the empty array and a key value pair is created in “gourps” where the key is the country name of that array object. In the second and subsequent iterations, the “groups” object is passed in to the callback and either a new key:value pair is created for the new array object in that iteration or the array object is added to an existing array in the “groups” object if that country has already been encountered in the loop. The result is an object where the keys are the country names and the values are arrays of all the data objects corresponding to that country from the input data.



```
// Function to create an object where data objects are stored
// by country.
function groupByCountry(data) {
    const newGroups = data.reduce(function(groups, item) {
    const group = (groups[item.location] || []);
    group.push(item);
    groups[item.location] = group;
    return groups;
    }, {});

    return newGroups;
}
```

Function to group data by country in an object

I utilised the same function in order to group the data by year that is used later. To do this I used the same function by adapted the key that is called on the object when creating the groups.



```
// Function to create an object where data objects are stored
// by year.
function groupByYear(data) {
    const newGroups = data.reduce(function(groups, item) {
    const group = (groups[item.Year] || []);
    group.push(item);
    groups[item.Year] = group;
    return groups;
    }, {});

    return newGroups;
}
```

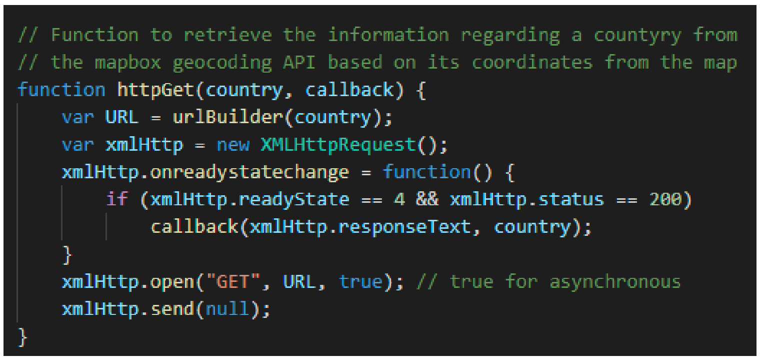
Function to group data by year in an object

4. On Click

Once this processing has completed, no more action is taken by the program until it receives input from the user when they click on the map. When they click on the map, the map.on(‘click’) function is triggered and a number of things happen.

4.1 httpGet

The httpGet function receives the coordinates of the country clicked on by the user and a callback function. It creates the URL to be searched using the urlBuilder and urlFormat functions and then undertakes an XMLHttpRequest using that URL. When the response data is received the callback function is triggered and the returned data passed in as an argument. The data that is returned is information about the country that the user clicked on from the mapbox forward geocoding API.

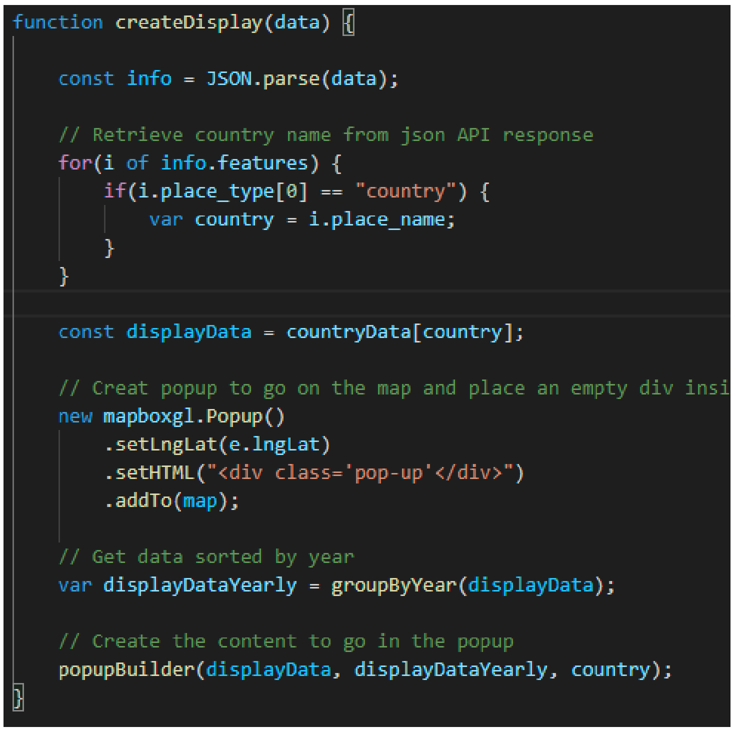


```
// Function to retrieve the information regarding a countyry from
// the mapbox geocoding API based on its coordinates from the map
function httpGet(country, callback) {
    var URL = urlBuilder(country);
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.onreadystatechange = function() {
        if (xmlHttp.readyState == 4 && xmlHttp.status == 200)
            callback(xmlHttp.responseText, country);
    }
    xmlHttp.open("GET", URL, true); // true for asynchronous
    xmlHttp.send(null);
}
```

httpGet function

4.2 createDispplay

The createDisplay callback function parses the retuned data as JSON and extracts from that the name of the country the user clicked on. This is then retrieved from the data that was process earlier by calling the data object using the country name as a key to return the data relating to that country. A Mapbox pop-up is then created and an empty div placed inside. The data is then grouped by year and the popupBuilder function is called.

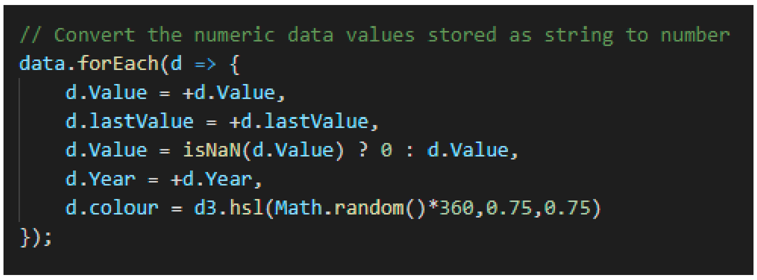


```
function createDisplay(data) {

    const info = JSON.parse(data);

    // Retrieve country name from json API response
    for(i of info.features) {
        if(i.place_type[0] == "country") {
            var country = i.place_name;
        }
    }

    const displayData = countryData[country];

    // Creat popup to go on the map and place an empty div insi
    new mapboxgl.Popup()
        .setLngLat(e.lngLat)
        .setHTML("<div class='pop-up'</div>")
        .addTo(map);

    // Get data sorted by year
    var displayDataYearly = groupByYear(displayData);

    // Create the content to go in the popup
    popupBuilder(displayData, displayDataYearly, country);
}
```

createDisplay function

5. popupBuilder

The popupBuilder function coordinates the creation of the visualisations that go in the pop-up. Firstly, the SVGs for both visualisations are created along with headings for them. Then, the numeric values in the data object that are currently stores as strings are converted to numbers and the pie chart is created.



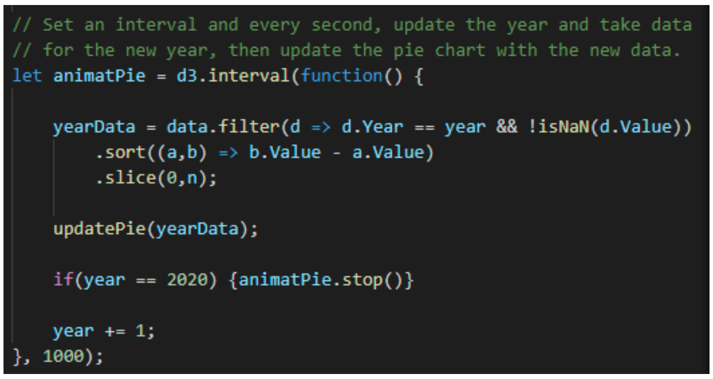
```
// Convert the numeric data values stored as string to number
data.forEach(d => {
    d.Value = +d.Value,
    d.lastValue = +d.lastValue,
    d.Value = isNaN(d.Value) ? 0 : d.Value,
    d.Year = +d.Year,
    d.colour = d3.hsl(Math.random()*360,0.75,0.75)
});
```

Convert string values to numbers

5.1 Pie Chart

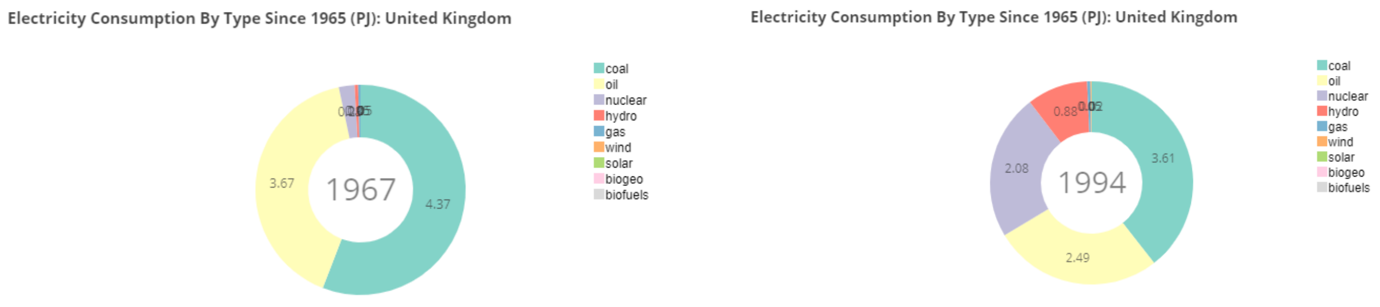
To create the pie chart, the data is split into arrays of data for each year, the first year being set to 1965. The year is then added to the centre of the pie to be displayed to the user. Each segment of the pie represents the amount of electricity that came from a particular source that was consumed by that country in the given year in petajoules.

The pie chart is created for the first year and then an interval function id initiated that cycles through the years from 1965 to 2020 at a rate of 1 year per second, In each new iteration of the interval function, a new years data is taken and the pie chart is updated using the updatePie function. The year is then incremented by 1. If the year equals 2020, the interval function stops.



```
// Set an interval and every second, update the year and take data
// for the new year, then update the pie chart with the new data.
let animatPie = d3.interval(function() {

    yearData = data.filter(d => d.Year == year && !isNaN(d.Value))
        .sort((a,b) => b.Value - a.Value)
        .slice(0,n);

    updatePie(yearData);

    if(year == 2020) {animatPie.stop()}

    year += 1;
}, 1000);
```

Interval function

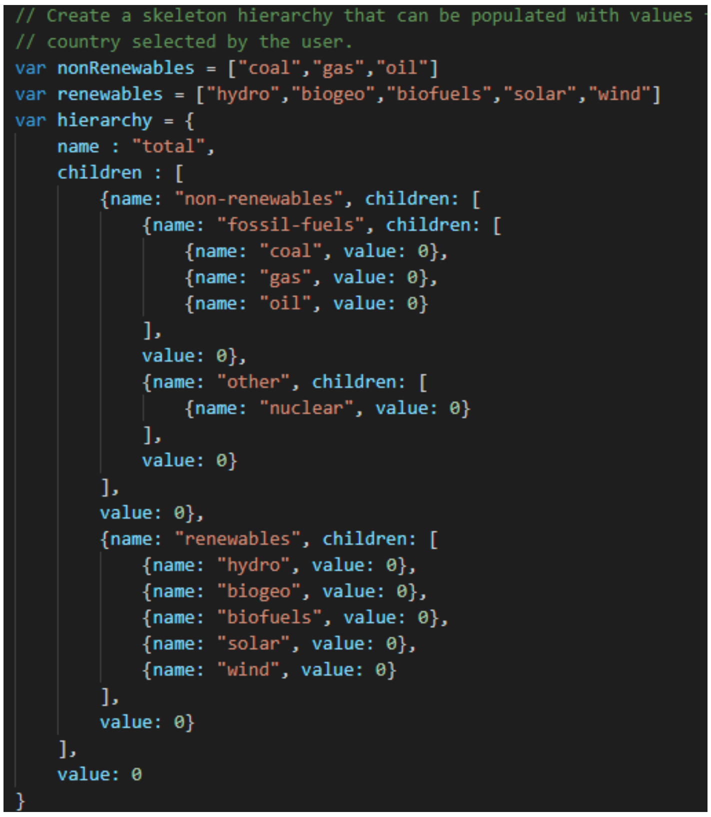


Pie chart for the United Kingdom during the interval function

5.2 Circle Chart

Along with the pie chart, I wanted to give the user a slightly more interactive visualisation that they can click around and explore. To do this I created a zoomable circle pack chart whereby there is a hierarchy of nested circles, and the user can click and zoom into these circles to see the detail. This chart was inspired by one created by Mike Bostok (Mike Bostok 2018)

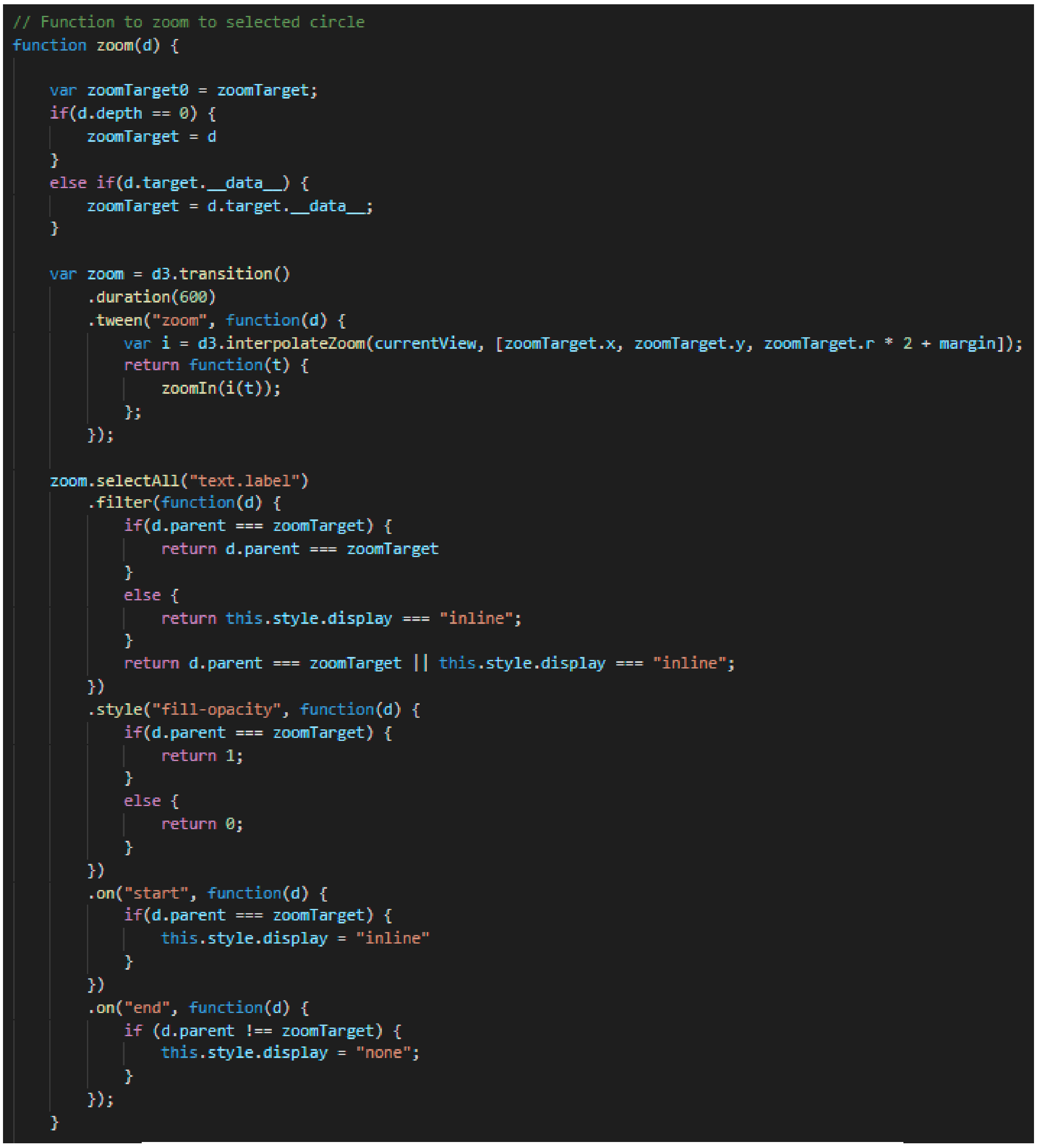
To do this I created a pack layout and a hierarchy with the data for 2020 for the given country. To convert the data to a hierarchy I created a skeleton object with values initiated to 0 that would be iterated over and populated with values based on the data used as an input. This hierarchy forms the basis of the circle chart.



```
// Create a skeleton hierarchy that can be populated with values
// country selected by the user.
var nonRenewables = ["coal","gas","oil"]
var renewables = ["hydro","biogeo","biofuels","solar","wind"]
var hierarchy = {
    name : "total",
    children : [
        {name: "non-renewables", children: [
            {name: "fossil-fuels", children: [
                {name: "coal", value: 0},
                {name: "gas", value: 0},
                {name: "oil", value: 0}
            ],
            value: 0},
            {name: "other", children: [
                {name: "nuclear", value: 0}
            ],
            value: 0}
        ],
        value: 0},
        {name: "renewables", children: [
            {name: "hydro", value: 0},
            {name: "biogeo", value: 0},
            {name: "biofuels", value: 0},
            {name: "solar", value: 0},
            {name: "wind", value: 0}
        ],
        value: 0}
    ],
    value: 0
}
```

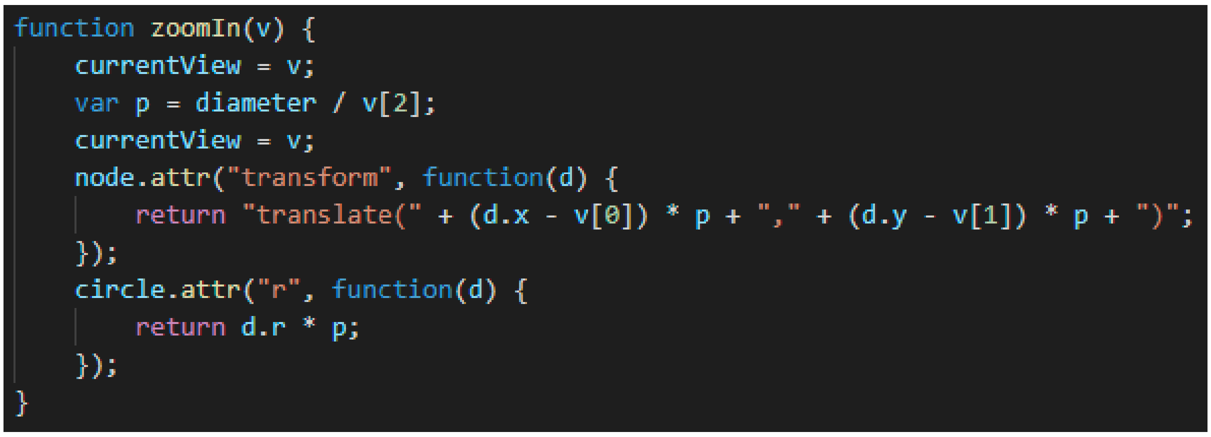
Hierarchy for the circle chart

I then calculated the values of the datapoints and their aggregated values e.g. the total of all renewable sources for the value for renewables. The, I created the circles, the text labels for the circles and 2 functions to control the zooming of the visualisation.



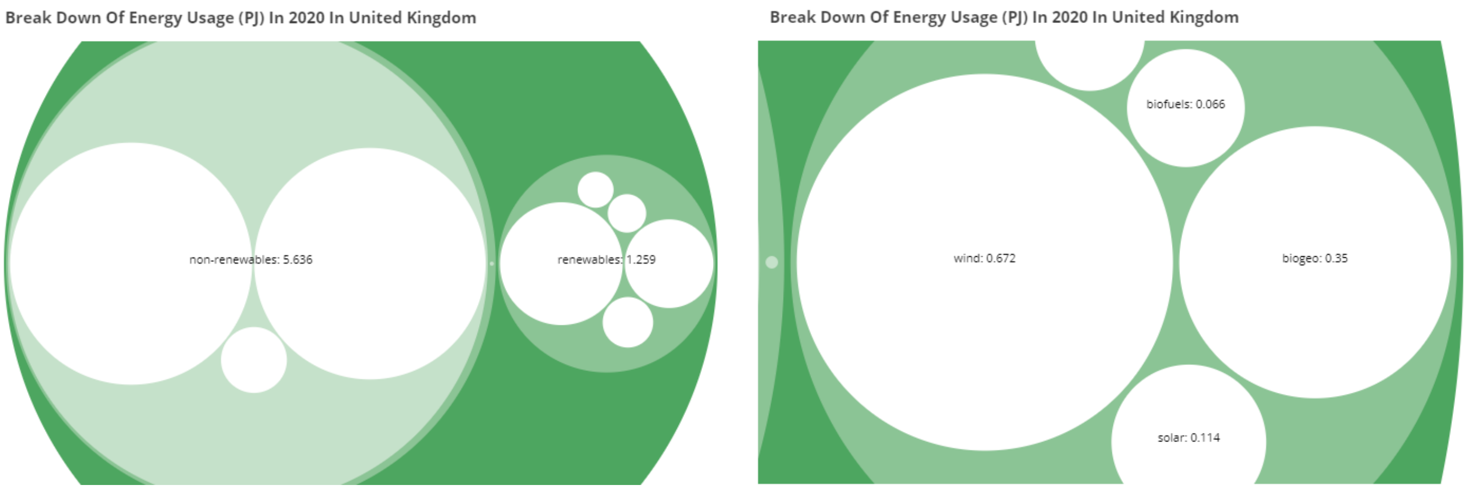
```
// Function to zoom to selected circle
function zoom(d) {

    var zoomTarget0 = zoomTarget;
    if(d.depth == 0) {
        zoomTarget = d
    }
    else if(d.target.__data__) {
        zoomTarget = d.target.__data__;
    }

    var zoom = d3.transition()
        .duration(600)
        .tween("zoom", function(d) {
            var i = d3.interpolateZoom(currentView, [zoomTarget.x, zoomTarget.y, zoomTarget.r * 2 + margin]);
            return function(t) {
                zoomIn(i(t));
            };
        });

    zoom.selectAll("text.label")
        .filter(function(d) {
            if(d.parent === zoomTarget) {
                return d.parent === zoomTarget
            }
            else {
                return this.style.display === "inline";
            }
            return d.parent === zoomTarget || this.style.display === "inline";
        })
        .style("fill-opacity", function(d) {
            if(d.parent === zoomTarget) {
                return 1;
            }
            else {
                return 0;
            }
        })
        .on("start", function(d) {
            if(d.parent === zoomTarget) {
                this.style.display = "inline"
            }
        })
        .on("end", function(d) {
            if (d.parent !== zoomTarget) {
                this.style.display = "none";
            }
        });
    }
```

Function to zoom to selected circle



```
function zoomIn(v) {
    currentView = v;
    var p = diameter / v[2];
    currentView = v;
    node.attr("transform", function(d) {
        return "translate(" + (d.x - v[0]) * p + "," + (d.y - v[1]) * p + ")";
    });
    circle.attr("r", function(d) {
        return d.r * p;
    });
}
```

Function to smoothly zoom to a new view of the chart using interpolated values



Example of the zoom in the circle chart when a user clicks on a circle

6. Conclusion

Overall, I am happy with the final output of this lab however given more time I would have included multiple data sources and tidied up the pop-up as some of the top and bottom of the circle chart has been cut off.

References

Mike Bostok (2018) “Zoomable Circle Packing”, Observable, https://observablehq.com/@d3/zoomable-circle-packing, Last Accessed: 08/04/2022