# CSCE 5320 Scientific Data Visualization

# ICE-8

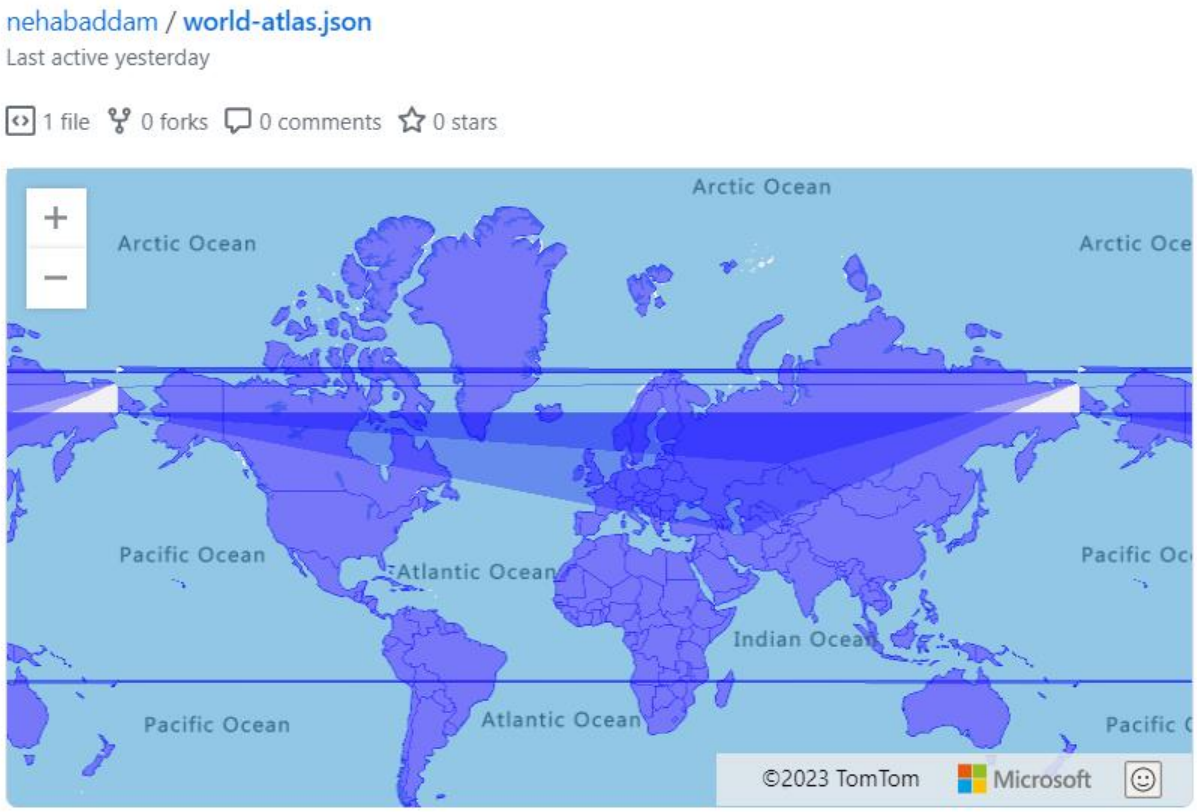
# Visualization of Spatial Data, Networks, and Trees

## 1. Making Maps(45 points)

Using TOPOJSON data links provided in the tutorial and making maps in d3.

### 1.1Create a world map with the world Topojson data. Submit the screenshots of your code (commented properly) with an explanation and provide the VizHub link to your code.

Firstly, I have loaded the TOPOJSON for the world map to the git hub gist, with the file name 'world-atlas.json'. This JSON file has topology information about all the continents and countries in the world.




Raw data is as shown below:

```
{"type":"Topology","objects":{"countries":{"type":"GeometryCollection","geometries":[{"type":"MultiPolygon","arcs":[[[0]],
[[1]]],"id":"242","properties":{"name":"Fiji"}},{"type":"Polygon","arcs":[[2,3,4,5,6,7,8,9,10]],"id":"834","properties":
{"name":"Tanzania"}},{"type":"Polygon","arcs":[[11,12,13,14]],"id":"732","properties":{"name":"W. Sahara"}},
{"type":"MultiPolygon","arcs":[[[15,16,17,18]],[[19]],[[20]],[[21]],[[22]],[[23]],[[24]],[[25]],[[26]],[[27]],[[28]],[[29]],
[[30]],[[31]],[[32]],[[33]],[[34]],[[35]],[[36]],[[37]],[[38]],[[39]],[[40]],[[41]],[[42]],[[43]],[[44]],[[45]],[[46]],
[[47]]],"id":"124","properties":{"name":"Canada"}},{"type":"MultiPolygon","arcs":[[[-19,48,49,50]],[[51]],[[52]],[[53]],
[[54]],[[55]],[[56]],[[57]],[[-17,58]],[[59]]],"id":"840","properties":{"name":"United States of America"}},
{"type":"Polygon","arcs":[[60,61,62,63,64,65]],"id":"398","properties":{"name":"Kazakhstan"}},{"type":"Polygon","arcs":
[[-63,66,67,68,69]],"id":"860","properties":{"name":"Uzbekistan"}},{"type":"MultiPolygon","arcs":[[[70,71]],[[72]],[[73]],
[[74]]],"id":"598","properties":{"name":"Papua New Guinea"}},{"type":"MultiPolygon","arcs":[[[-72,75]],[[76,77]],[[78]],
[[79,80]],[[81]],[[82]],[[83]],[[84]],[[85]],[[86]],[[87]],[[88]],[[89]]],"id":"360","properties":{"name":"Indonesia"}},
{"type":"MultiPolygon","arcs":[[[90,91]],[[92,93,94,95,96,97]]],"id":"032","properties":{"name":"Argentina"}},
{"type":"MultiPolygon","arcs":[[[-92,98]],[[99,-95,100,101]]],"id":"152","properties":{"name":"Chile"}},
{"type":"Polygon","arcs":[[-8,102,103,104,105,106,107,108,109,110,111]],"id":"180","properties":{"name":"Dem. Rep. Congo"}},
{"type":"Polygon","arcs":[[112,113,114,115]],"id":"706","properties":{"name":"Somalia"}},{"type":"Polygon","arcs":
[[-3,116,117,118,-113,119]],"id":"404","properties":{"name":"Kenya"}},{"type":"Polygon","arcs":
[[120,121,122,123,124,125,126,127]],"id":"729","properties":{"name":"Sudan"}},{"type":"Polygon","arcs":
[[-122,128,129,130,131]],"id":"148","properties":{"name":"Chad"}},{"type":"Polygon","arcs":
[[132,133]],"id":"332","properties":{"name":"Haiti"}},{"type":"Polygon","arcs":[[-133,134]],"id":"214","properties":
{"name":"Dominican Rep."}},{"type":"MultiPolygon","arcs":[[[135]],[[136]],[[137]],[[138]],[[139]],[[140]],[[141,142,143]],
[[144]],[[145]],[[146,147,148,149,-66,150,151,152,153,154,155,156,157,158,159,160,161]],[[162]],
[[163,164]]],"id":"643","properties":{"name":"Russia"}},{"type":"MultiPolygon","arcs":[[[165]],[[166]],
[[167]]],"id":"044","properties":{"name":"Bahamas"}},{"type":"Polygon","arcs":[[168]],"id":"238","properties":
{"name":"Falkland Is."}},{"type":"MultiPolygon","arcs":[[[169]],[[-161,170,171,172]],[[173]],
[[174]]],"id":"578","properties":{"name":"Norway"}},{"type":"Polygon","arcs":[[175]],"id":"304","properties":
{"name":"Greenland"}},{"type":"Polygon","arcs":[[176]],"id":"260","properties":{"name":"Fr. S. Antarctic Lands"}},
{"type":"Polygon","arcs":[[177,-77]],"id":"626","properties":{"name":"Timor-Leste"}},{"type":"Polygon","arcs":
[[178,179,180,181,182,183,184],[185]],"id":"710","properties":{"name":"South Africa"}},{"type":"Polygon","arcs":
[[-186]],"id":"426","properties":{"name":"Lesotho"}},{"type":"Polygon","arcs":
[[-50,186,187,188,189]],"id":"484","properties":{"name":"Mexico"}},{"type":"Polygon","arcs":
[[190,191,-93]],"id":"858","properties":{"name":"Uruguay"}},{"type":"Polygon","arcs":
[[-191,-98,192,193,194,195,196,197,198,199,200]],"id":"076","properties":{"name":"Brazil"}},{"type":"Polygon","arcs":
[[-194,201,-96,-100,202]],"id":"068","properties":{"name":"Bolivia"}},{"type":"Polygon","arcs":
[[-195,-203,-102,203,204,205]],"id":"604","properties":{"name":"Peru"}},{"type":"Polygon","arcs":
[[-196,-206,206,207,208,209,210]],"id":"170","properties":{"name":"Colombia"}},{"type":"Polygon","arcs":
[[-209,211,212,213]],"id":"591","properties":{"name":"Panama"}},{"type":"Polygon","arcs":
[[-213,214,215,216]],"id":"188","properties":{"name":"Costa Rica"}},{"type":"Polygon","arcs":
[[-216,217,218,219]],"id":"558","properties":{"name":"Nicaragua"}},{"type":"Polygon","arcs":
[[-219,220,221,222,223]],"id":"340","properties":{"name":"Honduras"}},{"type":"Polygon","arcs":
[[-222,224,225]],"id":"222","properties":{"name":"El Salvador"}},{"type":"Polygon","arcs":
[[-189,226,227,-223,-226,228]],"id":"320","properties":{"name":"Guatemala"}},{"type":"Polygon","arcs":
[[-188,229,-227]],"id":"084","properties":{"name":"Belize"}},{"type":"Polygon","arcs":
[[-197,-211,230,231]],"id":"862","properties":{"name":"Venezuela"}},{"type":"Polygon","arcs":
[[-198,-232,232,233]],"id":"328","properties":{"name":"Guyana"}},{"type":"Polygon","arcs":
[[-199,-234,234,235]],"id":"740","properties":{"name":"Suriname"}},{"type":"MultiPolygon","arcs":[[[-200,-236,236]],
```


Now I have written the code to display the world map on VizHub.

## index.html:

Firstly, we create an HTML file that will be used to display the world map. We use a link tag to link the styles.css file, which has CSS defined for the webpage. We also link the JavaScript files to make use of all the D3.js and TOPOJSON tools. We use the body tag to display the SVG image of the world map in the body, using the index.js file.

index.html

```html
<!DOCTYPE html to display the html page>
<html>

  <head>
    <title>ICE8 -  World Map</title>
    <link rel="stylesheet" href="styles.css">
    <script src="https://unpkg.com/d3@5.9.2/dist/d3.min.js"></script>
    <script src="https://unpkg.com/topojson@3.0.2/dist/topojson.min.js"></script>
  </head>
  <body>
    <svg width="960" height="500"></svg>
    <script src="bundle.js"></script>
  </body>
</html>
```

## index.js:

This file has the main code for creating the world map. Firstly, all the required functions and packages from topojson and d3 libraries are imported. Then we select the SVG element from the HTML code using the select function. We are also defining the height and width of the SVG image. We then create a header text to display the header, by appending it to the SVG element. The ‘geoNaturalEarth1’ function from D3.js is then called to create a projection for the world map, and the ‘geoPath’ function is used to create a path generator for the map. We are using the ‘scaleOrdinal’ function and the color scheme ‘schemePastel1’ to display each country in a different color, we are using a pastel color pallet here. We Append the path element to the SVG element  and determine the shape of the map as a sphere.

Finally, we load the JSON data from GitHub and convert it into a feature collection for different countries. For each country, path elements are created, class is set, and path data and color fill for each path are set based on countries.

index.js

```
//importing the json libraries and  topojson file
import { select, json, geoPath, geoNaturalEarth1} from 'd3';
import {feature} from 'topojson';

// for svg image
const svg = select('svg');

// height and width of the svg image
const width = +svg.attr('width');
const height = +svg.attr('height');

const HeaderText = "World Map";
svg.append('text').attr('y',40).attr('x',25).text(HeaderText).attr('class',"Header").attr('font-size', '20px');

//calling the geoNaturalEarth function
const projection = geoNaturalEarth1();
const pathGenerator = geoPath().projection(projection);

// create a color scale for the countries
const colorScale = d3.scaleOrdinal(d3.schemePastel1);

//for the path and shape for the world map
svg.append('path')
    .attr('class', 'sphere')
    .attr('d', pathGenerator({type: 'Sphere'}));

//below is the link for the json data with country names and the coordinates for the world map
json('https://gist.githubusercontent.com/nehabaddam/6cd52d69434b442213e9cc20c8610e9c/raw/bc011f4cb26ba285d9a78d0aed7389a7da31a25d/world-atlas.json')
    .then(data => {
      //for displaying the countries
      const countries = feature(data, data.objects.countries);

      // below lines are used for printing the world map with its respective boundaries and fill colors
      svg.selectAll('path')
        .data(countries.features)
          .enter().append('path')
          .attr('class', 'country')
          .attr('d', pathGenerator)
          .attr('fill', d => colorScale(d.properties.name))
    });

```

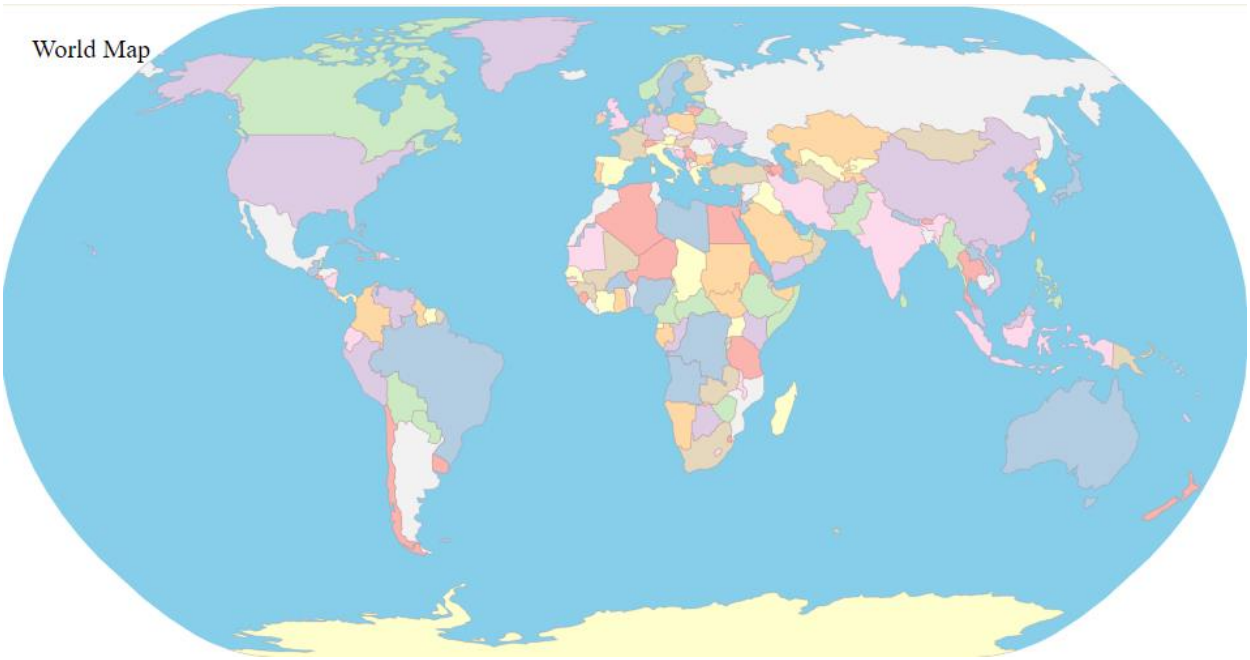
## styles.css

This file defines the style of the Webpage, by defining the background color and other parameters of the webpage.

styles.css

```
body {
  margin: 0px;
  overflow: hidden;
  background-color: white;
}

.sphere {
  fill: skyblue;
}

.country {
  stroke: brown;
  stroke-opacity: .15;

}
```

Below is the image of the world map.




**GitHub gist:**
https://gist.githubusercontent.com/nehabaddam/6cd52d69434b442213e9cc20c8610e9c/raw/1f2cf89618f7db82a2442abb7a94e11f70f418df/world-atlas.json

**VizHub Link:** https://vizhub.com/nehabaddam/ddbdba4659cc43aea66e32f3187b48c3

**VizHub ID:** nehabaddam

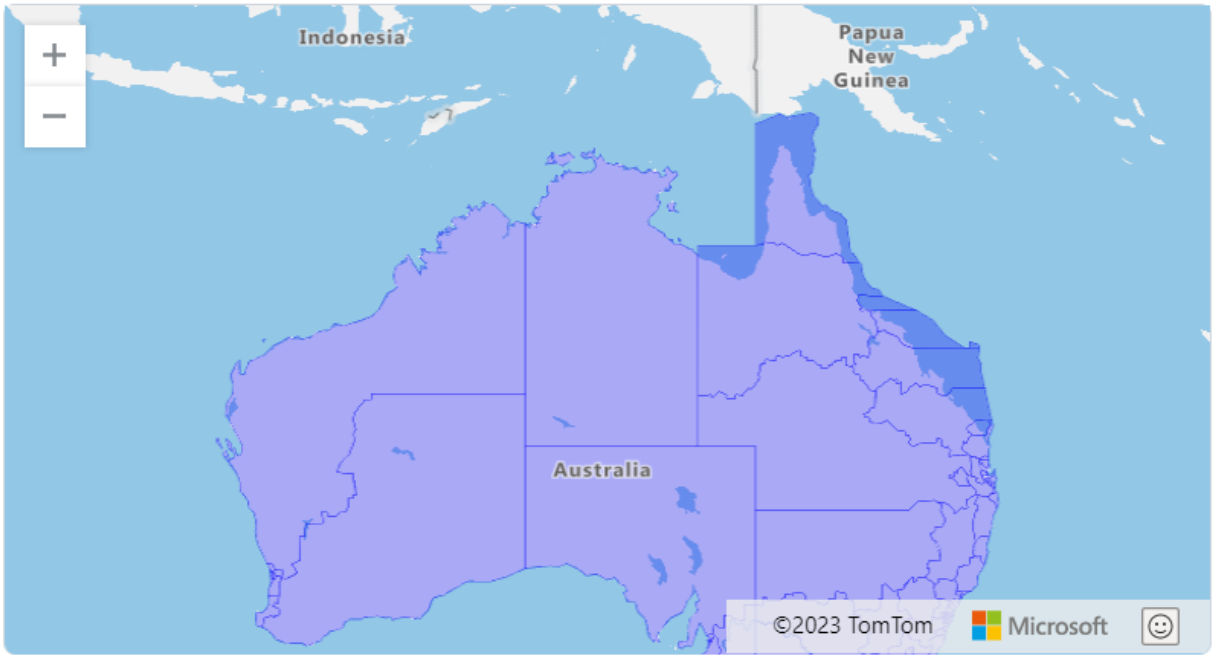
## 1.2 Create a country map with Topojson data. Any country (including the US) works. Submit the screenshots of your code (commented properly) with an explanation and provide the VizHub link to your code.

Firstly, I have loaded the TOPOJSON for the Australia country map to the git hub gist, with the file name country.json.

nehabaddam / **country.json**

Created yesterday

1 file 0 forks 0 comments 0 stars



{"type":"Topology","arcs":[[[68681,13361],[-48,-30]],[[68633,13331],[-153,66]],[[68480,13397],[82,9]],[[68562,13406],
[52,-4]],[[68614,13402],[67,-41]],[[68737,13427],[-24,-57]],[[68713,13370],[-32,-9]],[[68614,13402],[53,43]],[[68667,13445],
[70,-18]],[[68697,13589],[0,1]],[[68697,13590],[-2,-5]],[[68695,13585],[-2,0]],[[68693,13585],[0,-1]],[[68693,13584],[1,0]],
[[68694,13584],[1,-5]],[[68695,13579],[1,-1]],[[68696,13578],[1,0]],[[68697,13578],[2,-1]],[[68699,13577],[0,-1]],
[[68699,13576],[-8,2]],[[68691,13578],[-12,-38]],[[68679,13540],[-79,26]],[[68600,13566],[-18,-1]],[[68582,13565],[-5,79]],
[[68577,13644],[66,18]],[[68643,13662],[73,-54]],[[68716,13608],[-19,-19]],[[68877,13910],[-102,-61]],[[68775,13849],
[-41,-31]],[[68734,13818],[-13,-28]],[[68721,13790],[-78,-128]],[[68577,13644],[-21,-3]],[[68556,13641],[-160,215]],
[[68396,13856],[-22,239],[128,79]],[[68502,14174],[146,-141],[229,-123]],[[68570,13514],[-8,-108]],[[68480,13397],[-3,22]],
[[68477,13419],[22,83]],[[68499,13502],[13,26]],[[68512,13528],[43,22]],[[68555,13550],[15,-36]],[[68812,13608],[-96,0]],
[[68721,13790],[53,-155]],[[68774,13635],[38,-27]],[[67488,13487],[69,-97],[-158,-175],[-211,-61],[-36,-78],[-302,-13],
[-66,106],[-357,220],[-189,-44],[-99,133]],[[66139,13478],[-268,294],[18,101],[-170,96],[-96,-61],[-372,-46],[-26,462],
[125,108],[12,176],[-156,174],[-68,186],[24,169]],[[65162,15137],[195,84],[131,-52],[145,105],[-20,192],[283,224],[21,133],
[146,106],[170,-11],[-6,-254],[147,-19],[161,286],[402,-34],[57,85]],[[66994,15982],[192,-105],[146,-152],[324,-222]],
[[67656,15503],[-91,-72],[-14,-202],[95,-53],[76,-504],[303,-70]],[[68025,14602],[-105,-175],[67,-215],[-169,-178],
[-241,-17],[-100,-389],[11,-141]],[[68387,13601],[-119,29]],[[68268,13630],[12,136]],[[68280,13766],[49,37]],[[68329,13803],
[58,-202]],[[68810,13302],[-87,-76]],[[68723,13226],[-87,92]],[[68636,13318],[-3,13]],[[68713,13370],[97,-68]],
[[70939,16755],[-226,209],[-72,155],[35,103],[-406,-21],[-46,68]],[[70224,17269],[-74,194],[-2,180],[108,135],[-75,195],
[142,252]],[[70323,18225],[150,178],[185,136],[328,-36],[140,-98],[109,63]],[[71235,18468],[-149,-353],[-54,-299],[29,-190],
[93,-95],[-26,-179],[-101,-134],[-15,-224],[-73,-239]],[[68601,13089],[-178,-204],[-35,-246]],[[68388,12639],[-31,12]],
[[68357,12651],[-233,69],[71,143]],[[68195,12863],[166,106],[-22,108]],[[68339,13077],[80,70]],[[68419,13147],[182,-58]],
[[69086,14105],[-133,24],[-162,224],[-18,150]],[[68773,14503],[81,110],[142,-56],[25,-118],[144,-12]],[[69165,14427],
[67,-72]],[[69232,14355],[-146,-250]],[[67422,10498],[-4,-343],[-81,-113],[-1,-203],[-205,-521],[57,-240],[91,-157],
[-99,-88],[42,-304]],[[67222,8529],[-982,415],[-1271,555],[-76,71]],[[64893,9570],[105,199],[-227,319],[-51,435],
[-390,150]],[[64330,10673],[259,429],[-62,95]],[[64527,11197],[66,195],[-65,113],[62,132],[284,42],[122,126],[305,-60],
[139,81],[-57,265],[82,375],[85,116]],[[65550,12582],[245,-36],[240,-259],[7,-95],[176,-89],[336,-48],[99,27],[43,-284],
[469,-163],[153,311]],[[67318,11946],[193,-233],[-211,-210],[0,-102],[-131,-282],[-148,-193],[189,-352],[212,-76]],
[[66241,11675],[-100,111],[-202,-129]],[[65939,11657],[-192,-121],[-59,-258],[32,-276],[128,-35],[29,-181],[172,-89],
[66,158],[58,533],[76,95]],[[66249,11483],[-8,192]],[[64330,10673],[-170,-89],[-191,81],[-65,-121],[-272,5],[-153,-83],
[-228,10],[-403,167],[-151,-53]],[[62697,10590],[-278,-48]],[[62419,10542],[-342,110],[-145,-40],[-328,242],[-229,-82],
[-354,52],[-129,-46],[-60,-175],[-149,-184],[-281,116],[-221,192]],[[60181,10727],[-42,133],[-414,335],[-230,104],
[-151,126],[-132,-17],[-100,116],[1,155],[-212,13],[-97,307],[40,236],[-313,140],[-81,-12],[-350,145],[-84,-169],[-216,-54],
[-115,251],[-84,67],[-44,257],[-121,43],[-23,173],[-184,70],[-249,-9],[-218,57],[-237,-152],[-203,131],[-448,122]],
[[55874,13295],[-1,948]],[[55873,14243],[437,-52],[69,519],[216,-25],[-16,-124],[604,-65],[299,28],[137,161],[209,-16],
[-41,233],[542,-59],[26,194],[591,-82],[194,-168],[-42,-286],[437,-29],[10,63],[437,-54],[14,-161],[360,-38],[236,94],
[82,267],[391,4],[72,436],[210,-71],[432,-53],[-34,-376],[489,-18],[78,-131],[-50,-392],[52,-174]],[[62314,13868],
[-32,-206],[464,-56],[71,-171],[-44,-292],[224,-25],[237,-219],[-34,-219],[146,-16],[-86,-485],[-74,-119],[12,-183],
[-292,-84],[-220,-154],[-45,-213],[94,-115],[561,-143],[273,172],[498,-130],[460,-13]],[[68477,13419],[-63,-11]],
[[68414,13408],[-96,86]],[[68318,13494],[181,8]],[[68140,11788],[-137,-85]],[[68003,11703],[-144,-143],[9,-104],[-120,-235],
[-306,-484],[-20,-239]],[[67318,11946],[-18,224],[217,40],[239,150]],[[67756,12360],[339,128],[246,-22]],[[68341,12466],
[-41,-194],[-84,-66],[29,-231],[-142,-66],[37,-121]],[[68743,13433],[-6,-6]],[[68667,13445],[-1,32]],[[68666,13477],
[54,50]],[[68720,13527],[0,1]],[[68720,13528],[-2,8]],[[68718,13536],[47,-6]],[[68765,13530],[-22,-97]],[[68475,13624],
[-18,-42]],[[68457,13582],[-70,19]],[[68329,13803],[29,-9]],[[68358,13794],[117,-170]],[[68419,13147],[-43,170]],
[[68376,13317],[38,91]],[[68723,13226],[-122,-137]],[[68195,12863],[-538,-190],[99,-313]],[[65550,12582],[-127,180],
[116,218],[0,198],[154,171],[446,129]],[[67488,13487],[177,34],[182,-146],[180,139]],[[68027,13514],[137,-11]],
[[68164,13503],[14,-96]],[[68178,13407],[25,-307],[136,-23]],[[69355,14789],[-62,-220],[-128,-142]],[[68773,14503],

index.html:

Firstly, we create an HTML file that will be used to display the world map. We use a link tag to link the styles.css file, which has defined CSS for the webpage. Now we link JavaScript files to make use of all the D3.js and TopoJSON tools. We use the body tag to display the SVG image of the world map in the body, using the index.js file.

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>ICE8 - Country Map</title>
    <link rel="stylesheet" href="styles.css">
    <script src="https://unpkg.com/d3@5.9.2/dist/d3.min.js"></script>
    <script src="https://unpkg.com/topojson@3.0.2/dist/topojson.min.js">
    </script>

  </head>
  <body>
    <svg width="950" height="1000"></svg>
    <script src="bundle.js"></script>
  </body>
</html>
```

index.js:

This file has the main code for creating the Australia map. Firstly, all the required functions and packages from topojson and d3 libraries are imported i.e., select, JSON, CSV, geoPath, geoAlbersUsa, geoMercator, scaleOrdinal, and schemePastel1 from D3.js and feature, mesh from TopoJSON. By using the select function we get the SVG element from the HTML code. We are also defining the height and width of the SVG image. We then create a header text to display the header, by appending it to the SVG element, the header text displays ‘Australia Map’. The ‘geoMercator’ function from D3.js is then called to create a projection for the Australia map, and the ‘geoPath’ function is used to create a path generator for the map. We are using the ‘scaleOrdinal’ function and color scheme ‘schemePastel1’ to display different states in different colors, we are using a pastel color pallet. We then Append the path element to the SVG element and determine the shape of the map as a sphere.

Finally, we load the JSON data from GitHub and convert it into a feature collection for different countries. For each state, path elements are created, class is set, and path data and color fill for each path are set based on countries. On the whole, we use this code to generate a map of the country Australia with different state divisions colored in different shades.

index.js

```
import {select, json, csv, geoPath, geoAlbersUsa, geoMercator, scaleOrdinal, schemePastel1} from 'd3';
import { feature, mesh } from 'topojson';

// for svg image
const svg = select('svg');

const HeaderText = "Australia Map";
svg.append('text').attr('y', 40).attr('x', 350).text(HeaderText).attr('class', 'Header')
  .attr('font-family', 'Arial').attr('font-size', '50px').attr('fill', 'skyblue');



//The geoMercator function in d3.js is used to draw The spherical Mercator projection.
const projection = geoMercator()
  .translate([-900, -35])
  .scale([600]);

// given a JSON geometry object, to generates an SVG path
const pathGenerator = geoPath().projection(projection);

//Array of ten categorical colors which is returned as RGB hexadecimal strings.
const colorScale = scaleOrdinal(schemePastel1);

//loading data
json('https://gist.githubusercontent.com/nehabaddam/0be7470424c9533fcd9541a3dd4b9fde/raw/5ccdd1431b27918a919586dd11df9e61c3e43f6f/country.json')
    .then(data => {
  //extracting states data
  const countryShapes = feature(data,data.objects["2021_ELB_region"] );

countryShapes.features.shift()

  //Renders the path to a Canvas by appending path
  //generated by pathgenerator to svg
  svg
    .selectAll('path')
    .data(countryShapes.features)
    .enter()
    .append('path')
    .attr('class', 'states')
    .attr('fill', (d) => { return(colorScale(d.properties.Elect_div))})
    .attr('d', (d) => pathGenerator(d))
    .append('title')
    .text((d) => d.properties.NAME_1);

  //Rendering text with state names in the middle of state
  svg
    .selectAll('text')
    .data(countryShapes.features)
    .enter()
    .append('text')
    .text((d) => d.properties.NAME_1)
    .attr('x', d=>  pathGenerator.centroid(d)[0]   )
    .attr('y',d=>pathGenerator.centroid(d)[1])
    .attr("text-anchor", "middle")
    .attr("font-size", "5px");
});
```

## styles.css

This file defines the style of the Webpage, by defining the background color and other parameters of the webpage.

style.css

```css
body {
  margin: 0px;
  overflow: hidden;
}

.states {

  stroke: black;
  stroke-opacity: 0.5;
  stroke-width:2px
}

```

Below is the image of the country map of Australia.


**GitHub gist:**
https://gist.githubusercontent.com/nehabaddam/0be7470424c9533fcd9541a3dd4b9fde/raw/5ccdd1431b27918a919586dd11df9e61c3e43f6f/country.json

**VizHub Link:** https://vizhub.com/nehabaddam/ff8648db2be94bddb198fc126303b0a6

**VizHub ID:** nehabaddam

### 1.3Which geo-projection you used for the two maps above, explain the reason.

#### World Map:

I have used the **geoNaturalEarth** function for projecting the world map. The geoNaturalEarth1() function is a built-in function of D3 which is used to create a natural projection for the world map. This projection preserves both distance and direction from the central part of the map and a modified cylindrical projection for the outer regions. The reason for using **geoNaturalEarth** geo-projection is that it provides a good balance between the shape and size distortion of the countries and continents.

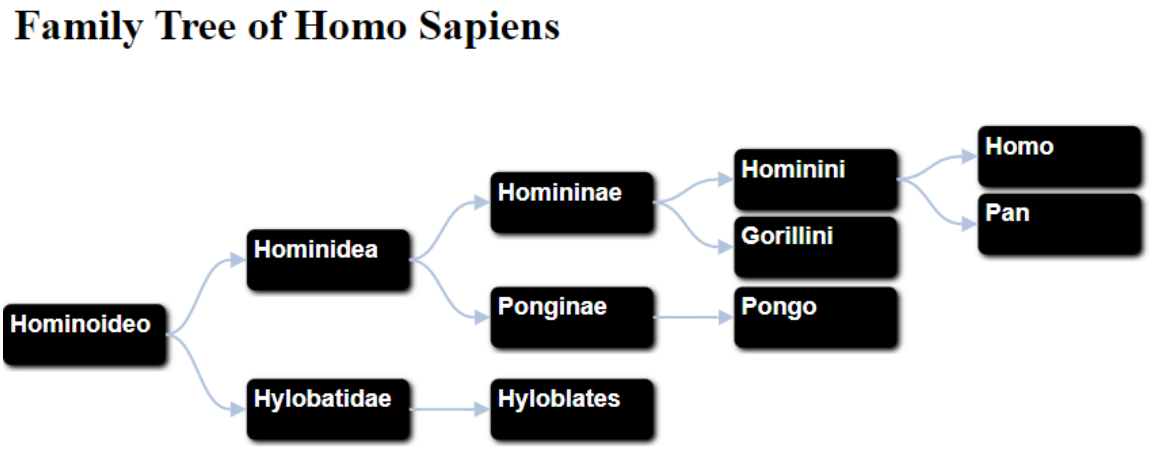
#### Country Map:

I have used the **geoMercator** function for projecting the country map of Australia. The **geoMercator**() function is a built-in function of D3 which is used to create a projection for the world map from the parameters given. This projection maps the Earth's surface onto a flat 2D plane. The reason for using **geoNaturalEarth** geo-projection is that it provides an easy way of representing the world map that is widely recognized. It preserves the shapes and angles of small features on the map, such as coastlines and country and continent borders, displaying detailed effectively.

## 2. Tree and Network (55 points)

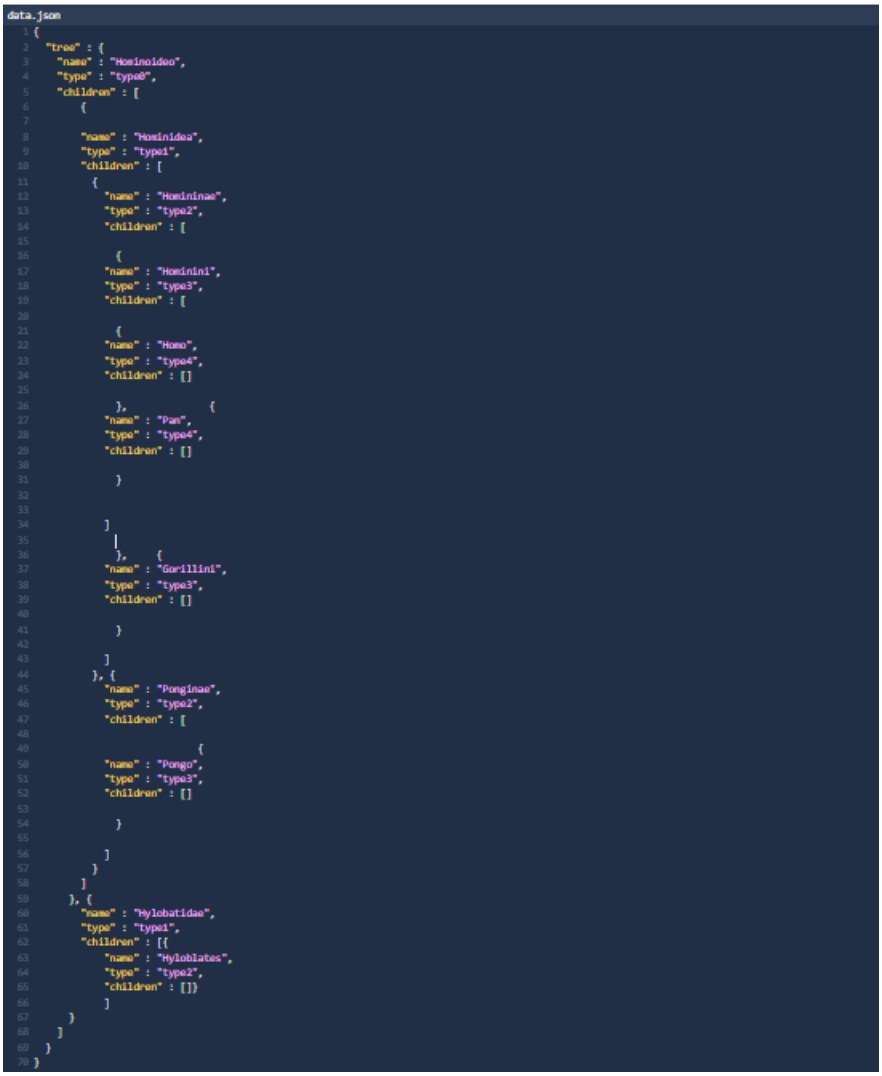
### 2.1Create a tree for animals or plant species (3 generations at least, with names) with d3.js. Submit the screenshots of your code (commented properly) with an explanation and provide the VizHub link to your code.

We are creating a Family Tree of Homo Sapiens. Initially I created a tree as shown below.



## data.json

The data file has tree structure data, that defines the parent-child relation between the data. Types are defined to display different colors at each level of the tree. For example, here Hominoideo is the parent of Hominidae.



```
data.json
{
  "tree" : {
    "name" : "Hominoideo",
    "type" : "type0",
    "children" : [
       {

       "name" : "Hominidea",
       "type" : "type1",
       "children" : [
        {
          "name" : "Homininae",
          "type" : "type2",
          "children" : [

           {
          "name" : "Hominini",
          "type" : "type3",
          "children" : [

           {
          "name" : "Homo",
          "type" : "type4",
          "children" : []

           },              {
          "name" : "Pan",
          "type" : "type4",
          "children" : []

           }


          ]
            |
           },    {
          "name" : "Gorillini",
          "type" : "type3",
          "children" : []

           }

          ]
        }, {
          "name" : "Ponginae",
          "type" : "type2",
          "children" : [

                        {
          "name" : "Pongo",
          "type" : "type3",
          "children" : []

           }

          ]
        }
       ]
      }, {
        "name" : "Hylobatidae",
        "type" : "type1",
        "children" : [{
          "name" : "Hyloblates",
          "type" : "type2",
          "children" : []}
          ]
     }
    ]
  }
}
```

## index.html

Firstly, we create an HTML file that will be used to display the world map. We use a link tag to link the styles.css file, which has defined CSS for the webpage. Now we link JavaScript files to make use of all the D3.js. We display a header and a container element with an ID of "tree-container" defined. We use the script tags to display the data.json file in the tree format, by creating the boxes.

index.html

```
<!DOCTYPE html><html><head>
    <meta charset="utf-8">
    <title>ICE-8 Family Tree</title>
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width">
    <link rel="stylesheet" type="text/css" href="styles.css">

    <script src="https://code.jquery.com/jquery-latest.min.js" type="text/javascript"></script>
    <script src="https://d3js.org/d3.v3.min.js" type="text/javascript"></script>
    <script src="index.js" type="text/javascript"></script>
  </head>
  <body>
    <h1>Family Tree of Homo Sapians </h1>
    <div class="container">
        <ct-visualization id="tree-container"></ct-visualization>
        <script>
            d3.json("data.json", function(error, json) {
                createBoxes(json.tree);
            });
        </script>
    </div>

<script src="bundle.js"></script></body></html>
```

## index.js

This code is used to create the boxes that represent each node. We code to display each node using a different color based on the types given in the JSON data file. If there is enough data to display the tree, we go ahead and generate a tree structure.

We first fix the root node and then create dimensions for the SVG image by calculating the depth and width of the tree. To the SVG, nodes are added, links are defined, and tooltips are defined. The functions make sure that all the nodes are represented clearly without any collision and the links also represent the parent-child relations correctly.

This code on the whole creates Boxes, that represent the nodes of the tree and colors the nodes based on the type of the node, and adds names to each node.

index.js

```
// CreateBoxes for background for each node in family tree
function createBoxes(jsonData) {
  // Varible to load tree
  var tree;

  // Varibles to use for creating links and boxes
  var baseSvg,svgGroup,nodeGroup, nodeGroupTooltip,linkGroup,linkGroupToolTip,defs;

  // width and height for the SVG
  var width = 850,
    height = 420;

  // rect for box element
  var boxNode = { width : 120, height : 45, textMargin : 5 },
    tooltip = { width : 150, height : 40, textMargin : 5 };

  // variable for transition duration
  var i = 0,
    duration = 600,
    root;

  // condition to check data available or not
    if (jsonData)
      drawFamilyTree(jsonData);
    else
    {
      alert('No data Mapped.');
    }

  // takes parameter data and creates familytree
  function drawFamilyTree(jsonData)
  {
    tree = d3.layout.tree().size([ height, width ]);
    root = jsonData;
    root.fixed = true;

    // adding colors to the boxes based on type
    var maxDepth = 0;
    var maxTreeWidth = breadthFirstTraversal(tree.nodes(root), function(currentLevel) {
      maxDepth++;
      currentLevel.forEach(function(node) {
        if (node.type == 'type0')
          node.color = 'teal';
        if (node.type == 'type1')
          node.color = 'olive';
        if (node.type == 'type2')
          node.color = 'tan';
        if (node.type == 'type3')
          node.color = 'pink';
        if (node.type == 'type4')
          node.color = 'gray';
        });
      });
    height = maxTreeWidth * (boxNode.height + 20) + tooltip.height + 20;
    width = maxDepth * (boxNode.width * 1.5) + tooltip.width / 2;

    tree = d3.layout.tree().size([ height, width ]);
    root.x0 = height / 2;
```

index.js

```
    root.y0 = 0;

    baseSvg = d3.select('#tree-container').append('svg')
      .attr('width', width)
    .attr('height', height + 120)
    .attr('class', 'svgContainer')

    svgGroup = baseSvg.append('g')
    .attr('class','drawarea')
    .append('g');

    nodeGroup = svgGroup.append('g')
          .attr('id', 'nodes');
    linkGroup = svgGroup.append('g')
          .attr('id', 'links');
    linkGroupToolTip = svgGroup.append('g')
                .attr('id', 'linksTooltips');
    nodeGroupTooltip = svgGroup.append('g')
                .attr('id', 'nodesTooltips');

    defs = baseSvg.append('defs');
    initAddArrows();
    initAddGraphics();

    updateLayout(root);
  }

  // updates the tree layout
  function updateLayout(source)
  {
    // Compute the new tree layout
    var nodes = tree.nodes(root).reverse(),
      links = tree.links(nodes);

    // Check if two nodes are in collision on the ordinates axe and move them
    breadthFirstTraversal(tree.nodes(root), collision);
    // Normalize for fixed-depth
    nodes.forEach(function(d) {
      d.y = d.depth * (boxNode.width * 1.5);
    });

  // ****************** Update the nodes ******************
    var node = nodeGroup.selectAll('g.node').data(nodes, function(d) {
      return d.id || (d.id = ++i);
    });
    var nodesTooltip = nodeGroupTooltip.selectAll('g').data(nodes, function(d) {
      return d.id || (d.id = ++i);
    });

    // Enter any new nodes at the parent's previous position
    // We use "insert" rather than "append", so when a new child node is added (after a click)
    // it is added at the top of the group, so it is drawed first
    // else the nodes tooltips are drawed before their children nodes and they
    // hide them
    var nodeEnter = node.enter().insert('g', 'g.node')
    .attr('class', 'node')
    .attr('transform', function(d) {
        return 'translate(' + source.y0 + ',' + source.x0 + ')'; })
```

index.js

```
    .on('click', function(d) {
            click(d);
      });
    var nodeEnterTooltip = nodesTooltip.enter().append('g')
      .attr('transform', function(d) {
          return 'translate(' + source.y0 + ',' + source.x0 + ')'; });

    nodeEnter.append('g').append('rect')
    .attr('rx', 6)
    .attr('ry', 6)
    .attr('width', boxNode.width)
    .attr('height', boxNode.height)
    .attr('class', 'node-rect')
    .attr('fill', function (d) { return d.color; })
    .attr('filter', 'url(#drop-shadow)');

    nodeEnter.append('foreignObject')
    .attr('x', boxNode.textMargin)
    .attr('y', boxNode.textMargin)
    .attr('width', function() {
          return (boxNode.width - boxNode.textMargin * 2) < 0 ? 0
              : (boxNode.width - boxNode.textMargin * 2)
        })
    .attr('height', function() {
          return (boxNode.height - boxNode.textMargin * 2) < 0 ? 0
              : (boxNode.height - boxNode.textMargin * 2)
        })
    .append('xhtml').html(function(d) {
          return '<div style="width: '
              + (boxNode.width - boxNode.textMargin * 2) + 'px; height: '
              + (boxNode.height - boxNode.textMargin * 2) + 'px;" class="node-text wordwrap">'
              + '<b>' + d.name + '</b><br><br>'
              + '</div>';
        })

    // Transition nodes to their new position.
    var nodeUpdate = node.transition().duration(duration)
    .attr('transform', function(d) { return 'translate(' + d.y + ',' + d.x + ')'; });

    // Transition exiting nodes to the parent's new position
    var nodeExit = node.exit().transition().duration(duration)
      .attr('transform', function(d) { return 'translate(' + source.y + ',' + source.x + ')'; })
      .remove();

  // ****************** Update the links ******************
    var link = linkGroup.selectAll('path').data(links, function(d) {
      return d.target.id;
    });

    d3.selection.prototype.moveToFront = function() {
        return this.each(function(){
            this.parentNode.appendChild(this);
          });
      };

    // Enter any new links at the parent's previous position.
      // Enter any new links at the parent's previous position.
      var linkenter = link.enter().insert('path', 'g')
```

index.js

```
    .attr('class', 'link')
    .attr('id', function(d) { return 'linkID' + d.target.id; })
    .attr('d', function(d) { return diagonalArrows(d); })
    .attr('marker-end', 'url(#end-arrow)');

  // Transition links to their new position.
  var linkUpdate = link.transition().duration(duration)
            .attr('d', function(d) { return diagonal(d); });

  // Stash the old positions for transition.
  nodes.forEach(function(d) {
    d.x0 = d.x;
    d.y0 = d.y;
  });
}

// function is processed on every node of a same level
// return the max level
  function breadthFirstTraversal(tree, func)
  {
    var max = 0;
    if (tree && tree.length > 0)
    {
      var currentDepth = tree[0].depth;
      var fifo = [];
      var currentLevel = [];

      fifo.push(tree[0]);
      while (fifo.length > 0) {
        var node = fifo.shift();
        if (node.depth > currentDepth) {
          func(currentLevel);
          currentDepth++;
          max = Math.max(max, currentLevel.length);
          currentLevel = [];
        }
        currentLevel.push(node);
        if (node.children) {
          for (var j = 0; j < node.children.length; j++) {
            fifo.push(node.children[j]);
          }
        }
      }
    func(currentLevel);
    return Math.max(max, currentLevel.length);
  }
  return 0;
  }

// x = ordoninates and y = abscissas
function collision(siblings) {
  var minPadding = 5;
  if (siblings) {
    for (var i = 0; i < siblings.length - 1; i++)
    {
      if (siblings[i + 1].x - (siblings[i].x + boxNode.height) < minPadding)
        siblings[i + 1].x = siblings[i].x + boxNode.height + minPadding;
    }
```

index.js

```
    }
  }

  function diagonalArrows(d) {
    var p0 = {
      x : d.source.x + boxNode.height / 2,
      y : (d.source.y + boxNode.width)
    }, p3 = {
      x : d.target.x + boxNode.height / 2,
      y : d.target.y  - 12 // -12, so the end arrows are just before the rect node
    }, m = (p0.y + p3.y) / 2, p = [ p0, {
      x : p0.x,
      y : m
    }, {
      x : p3.x,
      y : m
    }, p3 ];
    p = p.map(function(d) {
      return [ d.y, d.x ];
    });
    return 'M' + p[0] + 'C' + p[1] + ' ' + p[2] + ' ' + p[3];
  }

  function initAddGraphics() {
    var filter = defs.append("filter")
        .attr("id", "drop-shadow")
        .attr("color-interpolation-filters", "sRGB");

    filter.append("feOffset")
    .attr("result", "offOut")
    .attr("in", "SourceGraphic")
      .attr("dx", 0)
      .attr("dy", 0);

    filter.append("feGaussianBlur")
        .attr("stdDeviation", 2);

    filter.append("feOffset")
        .attr("dx", 2)
        .attr("dy", 2)
        .attr("result", "shadow");

    filter.append("feComposite")
      .attr("in", 'offOut')
      .attr("in2", 'shadow')
      .attr("operator", "over");
  }

  function initAddArrows() {
    // Build the arrows definitions
    // End arrow
    defs.append('marker')
    .attr('id', 'end-arrow')
    .attr('viewBox', '0 -5 10 10')
    .attr('refX', 0)
    .attr('refY', 0)
    .attr('markerWidth', 6)
    .attr('markerHeight', 6)
```

index.js

```
  }

  function initAddArrows() {
    // Build the arrows definitions
    // End arrow
    defs.append('marker')
    .attr('id', 'end-arrow')
    .attr('viewBox', '0 -5 10 10')
    .attr('refX', 0)
    .attr('refY', 0)
    .attr('markerWidth', 6)
    .attr('markerHeight', 6)
    .attr('orient', 'auto')
    .attr('class', 'arrow')
    .append('path')
    .attr('d', 'M0,-5L10,0L0,5');

    // End arrow selected
    defs.append('marker')
    .attr('id', 'end-arrow-selected')
    .attr('viewBox', '0 -5 10 10')
    .attr('refX', 0)
    .attr('refY', 0)
    .attr('markerWidth', 6)
    .attr('markerHeight', 6)
    .attr('orient', 'auto')
    .attr('class', 'arrowselected')
    .append('path')
    .attr('d', 'M0,-5L10,0L0,5');

    // Start arrow
    defs.append('marker')
    .attr('id', 'start-arrow')
    .attr('viewBox', '0 -5 10 10')
    .attr('refX', 0)
    .attr('refY', 0)
    .attr('markerWidth', 6)
    .attr('markerHeight', 6)
    .attr('orient', 'auto')
    .attr('class', 'arrow')
    .append('path')
    .attr('d', 'M10,-5L0,0L10,5');

    // Start arrow selected
    defs.append('marker')
    .attr('id', 'start-arrow-selected')
    .attr('viewBox', '0 -5 10 10')
    .attr('refX', 0)
    .attr('refY', 0)
    .attr('markerWidth', 6)
    .attr('markerHeight', 6)
    .attr('orient', 'auto')
    .attr('class', 'arrowselected')
    .append('path')
    .attr('d', 'M10,-5L0,0L10,5');
  }
}
```

## styles.css

This file defines the style of the Webpage, by defining the background color and other parameters for the box containing links, and arrows.

styles.css

```css
#tree-container {
  position: absolute;
  left: 0px;
  width: 100%;
}

.svgContainer {
  display: block;
    margin: auto;
}

.node {
  cursor: pointer;
}

.node-rect {
}

.node-rect-closed {
  stroke-width: 2px;
  stroke: rgb(0,0,0);
}

.link {
  fill: none;
  stroke: lightsteelblue;
  stroke-width: 2px;
}

.linkselected {
  fill: none;
  stroke: tomato;
  stroke-width: 2px;
}

.arrow {
  fill: lightsteelblue;
  stroke-width: 1px;
}

.arrowselected {
  fill: tomato;
  stroke-width: 2px;
}

.link text {
  font: 7px sans-serif;
  fill: #CC0000;
}

.wordwrap {
  white-space: pre-wrap; /* CSS3 */
  white-space: -moz-pre-wrap; /* Firefox */
  white-space: -pre-wrap; /* Opera <7 */
  white-space: -o-pre-wrap; /* Opera 7 */
  word-wrap: break-word; /* IE */
}
```

```
.node-text {
  font: 7px sans-serif;
  color: white;
}

.tooltip-text-container {
    height: 100%;
  width: 100%;
}

.tooltip-text {
  visibility: hidden;
  font: 7px sans-serif;
  color: white;
  display: block;
  padding: 5px;
}

.tooltip-box {
  background: rgba(0, 0, 0, 0.7);
  visibility: hidden;
  position: absolute;
  border-style: solid;
    border-width: 1px;
    border-color: black;
    border-top-right-radius: 0.5em;
}

p {
  display: inline;
}

.textcolored {
  color: orange;
}

a.exchangeName {
  color: orange;
}
```

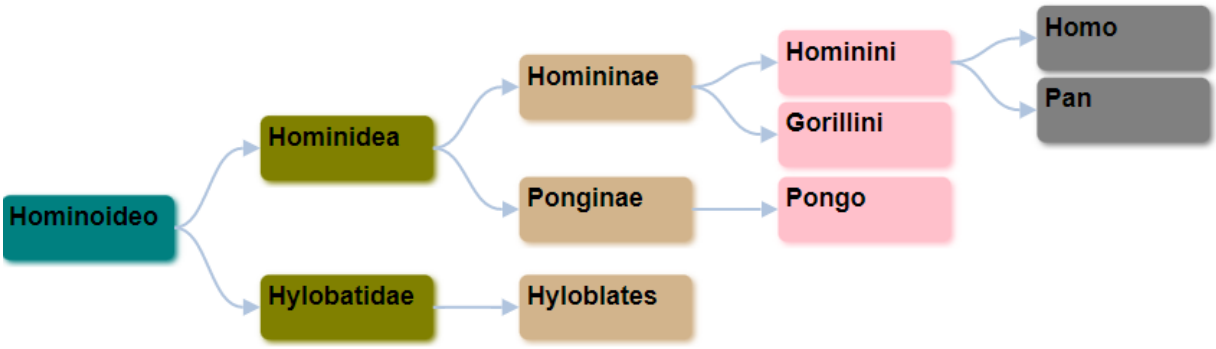
**VizHub Link:** https://vizhub.com/nehabaddam/0e08263c174d45c0b64cd1b0380dd482

**VizHub ID:** nehabaddam

## 2.2 Add elements to your tree, and use different colors OR shapes to represent different generations in your tree.

We have added type to the elements in the tree. Each level of the tree represents a generation. Each generation is represented using a different color. The root node is Hominoideo. The leaf nodes are the bottom nodes, for example, the Homo and Pan.

## Family Tree of Homo Sapiens




### 2.3Create a social network graph of you (5 people at least, with names) with d3.js. Submit the screenshots of your code (commented properly) with an explanation and provide the VizHub link to your code.

We are creating a Social Networking Graph with 10 people.

data.json

This data has Social Network Links between 10 friends, and it shows how they are related to each other.

data.json


```
{"links":
  [
    {"source": "Neha", "target": "Maharshi", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1},
    {"source": "Neha", "target": "SaiSree", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1},
    {"source": "Neha", "target": "Gowthami", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1},
    {"source": "Zeenath", "target": "SaiSree", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":5},
    {"source": "SaiSree", "target": "Ravallika", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1},
    {"source": "Ravallika", "target": "Tejas", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1},
    {"source": "Sathwika", "target": "SaiSree", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1},
    {"source": "Tejas", "target": "Karthik", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1},
    {"source": "Karthik", "target": "Sathwika", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1},
    {"source": "Akhila", "target": "Gowthami", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1},
     {"source": "Gowthami", "target": "Sathwika", "relationship": "Friend with", "srcType": "Person", "tgtType":"Friend", "value":1}
  ],
"nodes":
  [
    {"id": "Neha", "group":5},
    {"id": "Maharshi", "group":5},
    {"id": "Sathwika", "group":5},
    {"id": "SaiSree", "group":5},
    {"id": "Zeenath", "group":5},
    {"id": "Ravallika", "group":5},
    {"id": "Gowthami", "group":5},
    {"id": "Tejas", "group":5},
    {"id": "Karthik", "group":5},
    {"id": "Akhila", "group":5}
  ]}
```


## index.html

Firstly, we create an HTML file that will be used to display the world map. We use a link tag to link the styles.css file, which has defined CSS for the webpage. Now we link JavaScript files to make use of all the D3.js and TopoJSON tools. We use the body tag to display the SVG image of the social network in the body, using the index.js file.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Social Network Graph --> ICE-8</title>
    <link rel="stylesheet" href="styles.css">
    <script src="https://unpkg.com/d3@6/dist/d3.min.js">
    </script>
  </head>
  <body>
    <h1 style="text-align:center">Social Network Graph</h1>
    <svg width="1100" height="370"></svg>
    <script src="bundle.js"></script>
  </body>
</html>
```

## index.js

This file has the main code for creating the social network graph. Firstly, all the required functions and packages from topojson and d3 libraries are imported. Then we select the SVG

element from the HTML code using the select function. We are also defining the height and width of the SVG image and the color scale for the node. Then we define the node shape as a circle and markers are also defined.

Next, we load the JSON data from the data.json file and define the links and nodes. We are using the ‘forcesimulation’ function to create the links and nodes, events for nodes are defined using the ‘dragstarted’, ‘dragged’ and ‘dragended’ functions.

Finally, we create the links and nodes and set their attributes and events. The behavior of the simulation is set using the tick event.

index.js

```
import { select,selectAll,scaleOrdinal} from 'd3';

// defines all the variables that define the structure of the svg
const svg = select('svg');
const width = +svg.attr('width');
const height = +svg.attr('height');
const centre_x = width / 2;
const centre_y = height / 2;
const colorScale = ["pink"];

// set the structure of svg
svg.attr("preserveAspectRatio", "xMidYMid meet")
   .attr("viewBox", [0, 0, width * 2, height * 2]);

// define all nodes and markers
const defs = svg.append("defs");
    defs.selectAll("marker").data(["node"]).enter().append("marker")
        .attr("id", d => d).attr("viewBox", "0 -2.5 5 5").attr("refX", 25).attr("refY", 0).attr("markerWidth", 5).attr("markerHeight", 9)
        .attr("orient", "auto").append("path").attr("d", "M0,-2.5L5,0L0,2.5").attr("fill", "#999");

d3.json('data.json')
  .then(data => {

  // here we have defined links
  const links = data.links.map(d => Object.create(d));
  links.forEach((d, i) => {
    d.srcType = data.links[i].srcType;
    d.tgtType = data.links[i].tgtType;
    d.relationship = data.links[i].relationship;
    d.value = data.links[i].value;
  });

  //Below we have defined nodes
  const nodes = data.nodes.map(d => Object.create(d));
  nodes.forEach((d, i) => {
    d.group = data.nodes[i].group;
  });

  // links are created here.
  const simulation = d3.forceSimulation(nodes)
      .force("link", d3.forceLink(links)
             .id(d => d.id)
             .distance(120))
      .force("charge", d3.forceManyBody()
             .strength(-500)
            )
      .force("collision", d3.forceCollide()
              .radius(105)
            )
      .force("center", d3.forceCenter(width, height))
  ;

  //defined events
  const drag = simulation => {
    function dragstarted(event) {
      if (!event.active) simulation.alphaTarget(0.5).restart();
      event.subject.fx = event.subject.x;
      event.subject.fy = event.subject.y;
```

index.js

```
    }
    function dragged(event) {
      event.subject.fx = event.x;
      event.subject.fy = event.y;
    }
    function dragended(event, d) {
      if (!event.active) simulation.alphaTarget(0);
      event.subject.fx = null;
      event.subject.fy = null;
    }
    return d3.drag()
      .on("start", dragstarted)
      .on("drag", dragged)
      .on("end", dragended);
  }

  const link = svg.append("g")
  .selectAll("line")
    .data(links)
    .join("line")
      .attr("marker-end", "url(#node)");

  const node = svg.append("g")
    .attr("class", "node")
  .selectAll("g")
    .data(nodes)
    .enter()
    .append("g")
      .attr("transform", d => "translate(" + centre_x + ", " + centre_y + ")");
;

  const circles = node.append("circle")
      .attr("r", 75)
      .attr("fill", d => colorScale[0])
      .on("mouseover", focusNode)
      .on("mouseout", blurNode)
      .call(drag(simulation));

  const nodeLbl = node.append("text")
        .text(d => d.id)
        .attr("dy", "0.3em");

  simulation.on("tick", () => {
    link
      .attr("x1", d => d.source.x)
      .attr("y1", d => d.source.y)
      .attr("x2", d => d.target.x)
      .attr("y2", d => d.target.y);
    node
      .attr("transform", d => "translate(" + d.x + ", " + d.y + ")");
  });

  function focusNode() {
    d3.select(this).classed("nodeHover", true);
  }

  function blurNode() {
    d3.select(this).classed("nodeHover", false);
  }

});
```

## styles.css

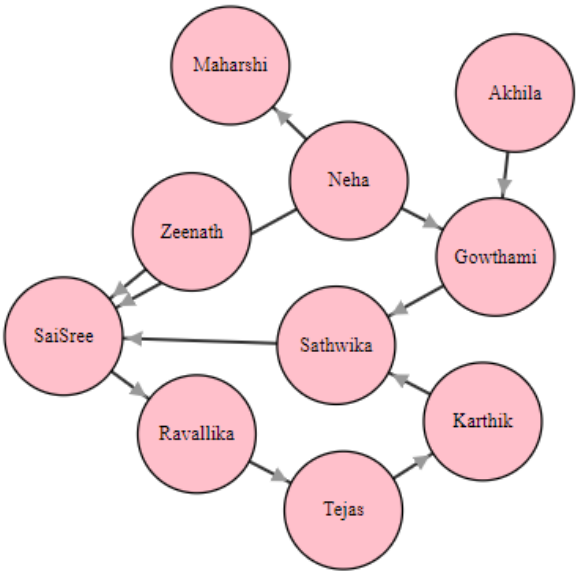
This file defines the style of the Webpage, by defining the background color and other parameters for the nodes and the links.

styles.css

```css
body {
  margin: 0px;
  overflow: hidden;
  font-family: bold;
  font-size: 25px;
}

line {
  stroke: #000;
  stroke-opacity: 0.8;
  stroke-width: 4;
}

text {
  fill: black;
  stroke: none;
  text-anchor: middle;
  pointer-events: none;
}

.node {
  stroke: black;
  stroke-width: 2.5;
  cursor: pointer;
}

line text {
  fill: #999;
  stroke: none;
}

.nodeHover {
  stroke: black;
  stroke-width: 15;
}

```

Below is the social network that is created:

# Social Network Graph




**VizHub Link:** https://vizhub.com/nehabaddam/00c793310f724bd4b85c48961375bc7a

**VizHub ID:** nehabaddam

## 2.4What are the differences between a network chart and a tree? What kind of data should be visualized in a network chart but not in a tree? Give examples.

| Network Chart | Tree Chart |
|---|---|
| It shows multiple relationships between elements, without any hierarchy. | It is a hierarchical data structure that offers a single parent-child relationship between elements. |
| It does not have any root node | It has the root node which is the top of the hierarchy. |
| Each element in a network can have multiple connections to other elements. | Each element in a tree can only have one parent. |
| The connections can be of different types, there is no specific direction. | There is a direction of flow from the root of the tree down to the leaves. |
| They are cyclic. | They are non-cyclic. |
| Examples: Social networks or communication networks, internet network | Example: Family tree, organizational trees, species trees |

We use network charts in visualizing the data that shows more than one connection between the elements, where there is no parent-child relation. For example, In a social network of friends, it shows the friends as nodes and connection between them is represented using a link. This relation can be better visualized as a network chart but not as a tree, because people can have multiple friends and those friends may have other friends that are friends of friends. This will form a mesh of friends. A network chart would be appropriate for a social networking chart like this, as we cannot form a tree structure with this kind of data. So, the data that forms a mesh of the network should be visualized using a network chart but not a tree.