PDF of Code

air-quality.js:

```
function AirQuality() {

  // Name for the visualisation for menu bar.
  this.name = 'Air Quality across G20 Countries';

  // Unique ID with no special characters.
  this.id = 'air-quality-g20';

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data.
  this.preload = function() {
  var self = this;
  this.data = loadTable(
    './data/air-quality/airQualityG20.csv', 'csv', 'header',
    // Callback function to set this.loaded to true.
    function(table) {
    self.loaded = true;
    });
  };

  this.setup = function() {
    // Check if data loaded
    if (!this.loaded) {
      console.log('Data not yet loaded');
      return;
    }

  // Create a select DOM element
  this.select = createSelect();
  this.select.position(400, 70);

  // Fill the options with various metrics of AQI
  var metrics = this.data.columns;

  // First entry is empty.
  for (let i = 1; i < metrics.length; i++) {
    this.select.option(metrics[i]);
    }
  };

  // Remove visual
```

```
this.destroy = function() {
   this.select.remove();
};

// Create a new bar chart object
this.bar = new BarChart(100, 450, 800, 200);

this.draw = function() {
   if (!this.loaded) {
      console.log('Data not yet loaded');
      return;
   }
   // Get the value of the selected metric
   var metricName = this.select.value();

   // Get the column of raw data for selected metric
   var col = this.data.getColumn(metricName);

   // Convert all data strings to numbers
   col = stringsToNumbers(col);

   // Copy the row labels from the table (the first item of each row).
   var labels = this.data.getColumn(0);

////// START OF MY CODE (ABOVE IS MODIFIED TECH DIV. RACE)//////

   // Create array of objects for colour-coded legend
   var legend = [];

   for (var i = 0; i < col.length; i++)
   {
      var cat = {};
      switch (true) {
         case (col[i] <= 50):
            cat = {
               "colour": '#00FF00',
               "category": 'Good'
            };
            break;
         case (col[i] <= 100):
            cat = {
               "colour": '#FFFF00',
               "category": 'Moderate'
            };
            break;
         case (col[i] <= 150):
            cat = {
```

```
                "colour": '#FF8000',
                "category": 'Unhealthy for Sensitive Groups'
              };
              break;
            case (col[i] <= 200):
              cat = {
                "colour": '#FF0000',
                "category": 'Unhealthy'
              };
              break;
            case (col[i] <= 300):
              cat = {
                "colour": '#660066',
                "category": 'Very Unhealthy'
              };
              break;
            default:
              cat = {
                "colour": '#660033',
                "category": 'Hazardous'
              };
              break;
          }
          legend.push(cat);
        }

        // Draw title
        var title = 'Air Quality Index in Nov 2022 by ' + metricName;

        // Draw the bar chart
        this.bar.draw(col, labels, legend, title);
      };
    }

////// END OF MY CODE //////
```

bar-chart.js:

```
///// START OF MY CODE /////

function BarChart(x, y, w, h) {

  this.x = x;
  this.y = y;
  this.width = w;
  this.height = h;

  this.draw = function(data, labels, legend, title) {

  // Check that data is not empty
    if (data.length == 0) {
      alert('Data has length zero!');
    }

    var barWidth = this.width / data.length;
    var colour;
    var category;
    var uniqueCategories = [];
    var count = 0;

    for (var i = 0; i < data.length; i++) {
      var maxData = max(data);
      var barHeight = map(data[i], 0, maxData, 0, this.height);
      var xPos = this.x + i * barWidth;
      var yPos = this.y - barHeight;

      // Get correct bar colour
      if (legend[i].colour) {
        colour = legend[i].colour;
      } else {
        colour = '#FFFFFF';
      }

      push();
      fill(colour);
      stroke(0);
      strokeWeight(1);

      // Draw bars
      rect(xPos, yPos, barWidth, barHeight);

      // Display data value on top of the bar
      textSize(14);
```

```
      fill(0);
      textAlign(CENTER);
      text(data[i], xPos + barWidth / 2, yPos - 20);

      // Display Labels diagonally with origin to the center of the label
      translate(xPos + barWidth / 2, this.y + 15);
      // Rotate by 45 degrees
      rotate(radians(45));
      textAlign(LEFT);
      // Draw the label
      text(labels[i], 0, 0);
      pop();

      // Check to see if legend item already exists, only add new items to the legend
      if (legend[i].category) {
        category = legend[i].category;
        if(!uniqueCategories.includes(category))
          {
            uniqueCategories.push(category);
            count++;
            this.makeLegendItem(category, count, colour);
          }
      }
    }
    // Get the title and draw it
    if (title) {
      noStroke();
      textAlign('center','center');
      textSize(22);
      text(title, this.x + this.width/2, 30);
    }

    // Calculate average for each metric selected
    var average = Math.round(getArrayAverage(data)*100)/100;
    // Set height of average line and Draw it
    var avgHeight = map(average, 0, maxData, 0, this.height);

    push();
    stroke(100);
    line(this.x, this.y - avgHeight, this.x + this.width, this.y - avgHeight);
    textSize(14);
    fill(100);
    text("Avg:", this.x - 30, this.y - avgHeight);
    text(average, this.x - 30, this.y - avgHeight + 20);
    pop();
};
```

```
// Method to draw unique colour coded legend item
   this.makeLegendItem = function(category, pos, colour) {
      var legendX = this.width - this.x;
      var legendY = 40 + pos * 20;

      fill(colour);
      rect(legendX, legendY, 15, 15);

      fill(0);
      noStroke();
      textAlign('left', 'center');
      textSize(12);
      text(category, legendX + 20, legendY + 8);
   };
}

///// END OF MY CODE /////
```

box.js:

```
function Box(x, y, width, height, category){

   var x = x;
   var y = y;
   var width = width;
   var height = height;

   this.category = category;

   this.mouseOver = function(mouseX, mouseY){
      if(mouseX > x && mouseX < x + width && mouseY > y && mouseY < y + height){
         return this.category.name;
      }
         return false;
   }

   this.draw =function(){
      fill(category.colour);
      rect(x, y, width, height);
   }
}
```

climate-change.js:

```
function ClimateChange() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Climate Change: Change in Temperature';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'climate-change';

  // Names for each axis.
  this.xAxisLabel = 'Year';
  this.yAxisLabel = '℃';

  var marginSize = 35;

  // Layout object to store all common plot layout parameters and
  // methods.
  this.layout = {
    marginSize: marginSize,

    // Locations of margin positions. Left and bottom have double margin
    // size due to axis and tick labels.
    leftMargin: marginSize * 2,
    rightMargin: width - marginSize,
    topMargin: marginSize,
    bottomMargin: height - marginSize * 2,
    pad: 5,

    plotWidth: function() {
      return this.rightMargin - this.leftMargin;
    },

    plotHeight: function() {
      return this.bottomMargin - this.topMargin;
    },

    // Boolean to enable/disable background grid.
    grid: false,

    // Number of axis tick labels to draw so that they are not drawn on
    // top of one another.
    numXTickLabels: 8,
    numYTickLabels: 8,
  };
```

```
// Property to represent whether data has been loaded.
this.loaded = false;

// Preload the data. This function is called automatically by the
// gallery when a visualisation is added.
this.preload = function() {
  var self = this;
  this.data = loadTable(
    './data/surface-temperature/surface-temperature.csv', 'csv', 'header',
    // Callback function to set the value
    // this.loaded to true.
    function(table) {
      self.loaded = true;
    });
};

this.setup = function() {
  // Font defaults.
  textSize(16);
  textAlign('center', 'center');

  // Set min and max years: assumes data is sorted by year.
  this.minYear = this.data.getNum(0, 'year');
  this.maxYear = this.data.getNum(this.data.getRowCount() - 1, 'year');

  // Find min and max temperature for mapping to canvas height.
  this.minTemperature = min(this.data.getColumn('temperature'));
  this.maxTemperature = max(this.data.getColumn('temperature'));

  // Find mean temperature to plot average marker.
  this.meanTemperature = mean(this.data.getColumn('temperature'));

  // Count the number of frames drawn since the visualisation
  // started so that we can animate the plot.
  this.frameCount = 0;

  // Create sliders to control start and end years. Default to
  // visualise full range.
  this.startSlider = createSlider(this.minYear,
                                  this.maxYear - 1,
                                  this.minYear,
                                  1);
  this.startSlider.position(400, 10);

  this.endSlider = createSlider(this.minYear + 1,
                                this.maxYear,
                                this.maxYear,
```

```
                        1);
  this.endSlider.position(600, 10);
};

this.destroy = function() {
  this.startSlider.remove();
  this.endSlider.remove();
};

this.draw = function() {
  if (!this.loaded) {
    console.log('Data not yet loaded');
    return;
  }

  // Prevent slider ranges overlapping.
  if (this.startSlider.value() >= this.endSlider.value()) {
    this.startSlider.value(this.endSlider.value() - 1);
  }
  this.startYear = this.startSlider.value();
  this.endYear = this.endSlider.value();

  // Draw title
  this.drawTitle();

  // Draw all y-axis tick labels.
  drawYAxisTickLabels(this.minTemperature,
                      this.maxTemperature,
                      this.layout,
                      this.mapTemperatureToHeight.bind(this),
                      1);

  // Draw x and y axis.
  drawAxis(this.layout);

  // Draw x and y axis labels.
  drawAxisLabels(this.xAxisLabel,
                 this.yAxisLabel,
                 this.layout);

  // Plot average line.
  stroke(200);
  strokeWeight(1);
  line(this.layout.leftMargin,
       this.mapTemperatureToHeight(this.meanTemperature),
       this.layout.rightMargin,
       this.mapTemperatureToHeight(this.meanTemperature));
```

```
// Plot all temperatures between startYear and endYear using the
// width of the canvas minus margins.
var previous;
var numYears = this.endYear - this.startYear;
var segmentWidth = this.layout.plotWidth() / numYears;

// Count the number of years plotted each frame to create
// animation effect.
var yearCount = 0;

// Loop over all rows but only plot those in range.
for (var i = 0; i < this.data.getRowCount(); i++) {

 // Create an object to store data for the current year.
 var current = {
  // Convert strings to numbers.
  'year': this.data.getNum(i, 'year'),
  'temperature': this.data.getNum(i, 'temperature')
 };

 if (previous != null
    && current.year > this.startYear
    && current.year <= this.endYear) {

  // Draw background gradient to represent colour temperature of
  // the current year.
  noStroke();
  fill(this.mapTemperatureToColour(current.temperature));
  rect(this.mapYearToWidth(previous.year),
      this.layout.topMargin,
      segmentWidth,
      this.layout.plotHeight());

  // Draw line segment connecting previous year to current
  // year temperature.
  stroke(0);
  line(this.mapYearToWidth(previous.year),
      this.mapTemperatureToHeight(previous.temperature),
      this.mapYearToWidth(current.year),
      this.mapTemperatureToHeight(current.temperature));

  // The number of x-axis labels to skip so that only
  // numXTickLabels are drawn.
  var xLabelSkip = ceil(numYears / this.layout.numXTickLabels);

  // Draw the tick label marking the start of the previous year.
```

```
    if (yearCount % xLabelSkip == 0) {
      drawXAxisTickLabel(previous.year, this.layout,
                         this.mapYearToWidth.bind(this));
    }

    // When six or fewer years are displayed also draw the final
    // year x tick label.
    if ((numYears <= 6
        && yearCount == numYears - 1)) {
      drawXAxisTickLabel(current.year, this.layout,
                         this.mapYearToWidth.bind(this));
    }

    yearCount++;
  }

  // Stop drawing this frame when the number of years drawn is
  // equal to the frame count. This creates the animated effect
  // over successive frames.
  if (yearCount >= this.frameCount) {
    break;
  }

  // Assign current year to previous year so that it is available
  // during the next iteration of this loop to give us the start
  // position of the next line segment.
  previous = current;
 }

 // Count the number of frames since this visualisation
 // started. This is used in creating the animation effect and to
 // stop the main p5 draw loop when all years have been drawn.
 this.frameCount++;

 // Stop animation when all years have been drawn.
 if (this.frameCount >= numYears) {
  //noLoop();
 }
};

this.mapYearToWidth = function(value) {
 return map(value,
            this.startYear,
            this.endYear,
            this.layout.leftMargin,   // Draw left-to-right from margin.
            this.layout.rightMargin);
};
```

```
  this.mapTemperatureToHeight = function(value) {
    return map(value,
               this.minTemperature,
               this.maxTemperature,
               this.layout.bottomMargin, // Lower temperature at bottom.
               this.layout.topMargin);   // Higher temperature at top.
  };

  this.mapTemperatureToColour = function(value) {
    var red =  map(value,
                   this.minTemperature,
                   this.maxTemperature,
                   0,
                   255);
    var blue = 255 - red;
    return color(red, 0, blue, 100);
  };

    this.drawTitle = function() {
       push();
       fill(0);
       noStroke();
       textSize(24);
       textAlign(CENTER,CENTER);
       text(this.name,width/2 + 200, 20);
       pop();
    };
}
```

eating-habits.js:

```
function EatingHabits() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Eating Habits Survey';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'eating-habits';

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
 this.preload = function() {
  var self = this;
  this.data = loadTable(
   './data/food/finalData.csv', 'csv', 'header',
   // Callback function to set the value
   // this.loaded to true.
   function(table) {
    self.loaded = true;
   });

 };

  this.setup = function() {
     if (!this.loaded){
        console.log('Data not yet loaded');
        return;
     }

     this.waffles = [];

     // Fill with days
     this.days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
                  "Sunday"];

     // Fill the values
     this.values = ['Take-away', 'Cooked from fresh', 'Ready meal', 'Ate out',
                  'Skipped meal', 'Left overs'];

     // fill waffles array with waffles

     for(var i = 0; i < this.days.length; i++){
```

```
        if(i < 4){
          this.waffles.push(new Waffle(100 + (i*220), 90, 200, 200, 10, 10, this.data,
this.days[i], this.values, this.days[i]));
        } else {
          this.waffles.push(new Waffle(100 + ((i-4)*220), 330, 200, 200, 10, 10, this.data,
this.days[i], this.values, this.days[i]));
        }
      }
  };

  this.destroy = function() {
  };

  this.draw = function() {


    // Draw a title
    this.drawTitle();

    // Draw the Waffles
    for(var i = 0; i < this.waffles.length; i++){
      this.waffles[i].draw();
    }

    for(var i = 0; i < this.waffles.length; i++){
      this.waffles[i].checkMouse(mouseX, mouseY);
    }
  };

  this.drawTitle = function() {
    fill(0);
    noStroke();
    textSize(22);
    textAlign(CENTER,CENTER);
    text(this.name,width/2, 20);
  };

}
```

gallery.js:

```
function Gallery() {

  this.visuals = [];
  this.selectedVisual = null;
  var self = this;


  // Add a new visualisation to the navigation bar.
  this.addVisual = function(vis) {

    // Check that the vis object has an id and name.
    if (!vis.hasOwnProperty('id')
        && !vis.hasOwnProperty('name')) {
      alert('Make sure your visualisation has an id and name!');
    }

    // Check that the vis object has a unique id.
    if (this.findVisIndex(vis.id) != null) {
      alert(`Vis '${vis.name}' has a duplicate id: '${vis.id}'`);
    }

    this.visuals.push(vis);


    // Create menu item.
    var menuItem = createElement('li', vis.name);
    menuItem.addClass('menu-item');
    menuItem.id(vis.id);

    menuItem.mouseOver(function(e)
    {

       var el = select('#' + e.srcElement.id);
       el.addClass("hover");
    })

    menuItem.mouseOut(function(e)
    {
       var el = select('#' + e.srcElement.id);
       el.removeClass("hover");
    })

    menuItem.mouseClicked(function(e)
    {
       //remove selected class from any other menu-items
```

```
    var menuItems = selectAll('.menu-item');

    for(var i = 0; i < menuItems.length; i++)
    {
       menuItems[i].removeClass('selected');
    }

    var el = select('#' + e.srcElement.id);
    el.addClass('selected');

    self.selectVisual(e.srcElement.id);

  })


  var visMenu = select('#visuals-menu');
  visMenu.child(menuItem);

  // Preload data if necessary.
  if (vis.hasOwnProperty('preload')) {
    vis.preload();
  }
};

this.findVisIndex = function(visId) {
  // Search through the visualisations looking for one with the id
  // matching visId.
  for (var i = 0; i < this.visuals.length; i++) {
    if (this.visuals[i].id == visId) {
      return i;
    }
  }

  // Visualisation not found.
  return null;
};

this.selectVisual = function(visId){
  var visIndex = this.findVisIndex(visId);

  if (visIndex != null) {
    // If the current visualisation has a deselect method run it.
    if (this.selectedVisual != null
        && this.selectedVisual.hasOwnProperty('destroy')) {
      this.selectedVisual.destroy();
    }
```

```
    // Select the visualisation in the gallery.
    this.selectedVisual = this.visuals[visIndex];

    // Initialise visualisation if necessary.
    if (this.selectedVisual.hasOwnProperty('setup')) {
      this.selectedVisual.setup();
    }

    // Enable animation in case it has been paused by the current
    // visualisation.
    loop();
  }
 };
}
```

heatmap.js:

///// START OF MY CODE /////

```
function HeatMap(xCoord, yCoord, zoom) {

   // Set coordinates of base map layer and zoom level
   this.xCoord = xCoord;
   this.yCoord = yCoord;
   this.zoom = zoom;

   // Creat a map div DOM element.
   this.mapDiv = createDiv();
   this.mapDiv.id('map');
   this.mapDiv.position(400, 80);

   // Set map properties using constructor properties
   this.map = L.map('map').setView([this.xCoord, this.yCoord],this.zoom);


   // Embed the map tile layer from OpenStreetMap.
   // https://leafletjs.com/examples/quick-start/
   var baseLayer = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
   maxZoom: 19,
   attribution: '&copy; <a
href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
   }).addTo(this.map);
```

///// WORK IN PROGRESS /////

```
   // Configuration Variable
   var cfg = {
   // radius should be small ONLY if scaleRadius is true (or small radius is intended)
   // if scaleRadius is false it will be the constant radius used in pixels
   "radius": 2,
   "maxOpacity": .8,
   // scales the radius based on map zoom
   "scaleRadius": true,
   // if set to false the heatmap uses the global maximum for colorization
   // if activated: uses the data maximum within the current map boundaries
   //   (there will always be a red spot with useLocalExtremas true)
   "useLocalExtrema": true,
   // which field name represents the latitude
   latField: 'lat',
   // which field name represents the longitude
   lngField: 'lng',
   // which field name in your data represents the data value - default "value"
```

```
    valueField: 'count'
    };

    // Test data for heat map
    var testData = {
        max: 8,
        data: [{lat: 54.7023545, lng:-3.2765753, count: 3},{lat: 49.0, lng:-3.5, count: 1}]
        };

///// WORK IN PROGRESS /////

    // Method to remove the map when another visual selected
    this.mapDestroy = function(){
        this.map.remove();
        this.mapDiv.remove();
    };
}

///// END OF MY CODE /////
```

helper-function.js:

```
// ---------------------------------------------------------------------
// Data processing helper functions.
// ---------------------------------------------------------------------
function sum(data) {
  var total = 0;

  // Ensure that data contains numbers and not strings.
  data = stringsToNumbers(data);

  for (let i = 0; i < data.length; i++) {
    total = total + data[i];
  }

  return total;
}

function mean(data) {
  var total = sum(data);

  return total / data.length;
}

function sliceRowNumbers (row, start=0, end) {
  var rowData = [];

  if (!end) {
    // Parse all values until the end of the row.
    end = row.arr.length;
  }

  for (i = start; i < end; i++) {
    rowData.push(row.getNum(i));
  }

  return rowData;
}

function stringsToNumbers (array) {
  return array.map(Number);
}

// ---------------------------------------------------------------------
// Plotting helper functions
// ---------------------------------------------------------------------
```

```
function drawAxis(layout, colour=0) {
  stroke(color(colour));

  // x-axis
  line(layout.leftMargin,
       layout.bottomMargin,
       layout.rightMargin,
       layout.bottomMargin);

  // y-axis
  line(layout.leftMargin,
       layout.topMargin,
       layout.leftMargin,
       layout.bottomMargin);
}

function drawAxisLabels(xLabel, yLabel, layout) {
  fill(0);
  noStroke();
  textAlign('center', 'center');

  // Draw x-axis label.
  text(xLabel,
       (layout.plotWidth() / 2) + layout.leftMargin,
       layout.bottomMargin + (layout.marginSize * 1.5));

  // Draw y-axis label.
  push();
  translate(layout.leftMargin - (layout.marginSize * 1.5),
            layout.bottomMargin / 2);
  rotate(- PI / 2);
  text(yLabel, 0, 0);
  pop();
}

function drawYAxisTickLabels(min, max, layout, mapFunction,
                             decimalPlaces) {
  // Map function must be passed with .bind(this).
  var range = max - min;
  var yTickStep = range / layout.numYTickLabels;

  fill(0);
  noStroke();
  textAlign('right', 'center');

  // Draw all axis tick labels and grid lines.
  for (i = 0; i <= layout.numYTickLabels; i++) {
```

```
    var value = min + (i * yTickStep);
    var y = mapFunction(value);

    // Add tick label.
    text(value.toFixed(decimalPlaces),
        layout.leftMargin - layout.pad,
        y);

    if (layout.grid) {
      // Add grid line.
      stroke(200);
      line(layout.leftMargin, y, layout.rightMargin, y);
    }
  }
}

function drawXAxisTickLabel(value, layout, mapFunction) {
  // Map function must be passed with .bind(this).
  var x = mapFunction(value);

  fill(0);
  noStroke();
  textAlign('center', 'center');

  // Add tick label.
  text(value,
       x,
       layout.bottomMargin + layout.marginSize / 2);

  if (layout.grid) {
    // Add grid line.
    stroke(220);
    line(x,
         layout.topMargin,
         x,
         layout.bottomMargin);
  }
}



// function to generate Random ID
function getRandomID(){
    var alpha = "abcdefghijklmnopqrstuvwxyz0123456789";
    var s = "";
    for(var i = 0; i < 10;i++){
        s += alpha[floor(random(0, alpha.length))];
```

```
    }

    return s;
}

///// START OF MY CODE /////

// This function calculates the the average of an array fo numbers to 2 decimal points.
function getArrayAverage(arr){
    var sum = 0;
    for(var i = 0; i < arr.length; i++){
        sum += arr[i];
    }
    var avg = (sum/arr.length).toFixed(2);
    return avg;
}


// This function uses an algortihm to calculate the median of an array of numbers
function getArrayMedian(arr){
    // Check if arr is an array
    if (!Array.isArray(arr)) {
        console.error("Error: Input is not an array.");
        return undefined; // or handle the error in an appropriate way
    }

    // Using this comparator function, we can assure expected result
    arr.sort((a, b) => a - b);
    var middleIndex = Math.floor(arr.length / 2);

    // Check if array is of even length, return average of 2 middle elements
    if (arr.length % 2 == 0){
        return (arr[middleIndex - 1] + arr[middleIndex]) / 2;
    }
    // Else return the middle item
    else{
        return arr[middleIndex];
    }
}

///// END OF MY CODE /////
```

index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Case study 2: Data visualisation</title>

    <!-- Libraries -->
    <script src="lib/p5.min.js"></script>
    <script src="lib/p5.dom.min.js"></script>
    <!-- HeatMap -->
    <script src="https://unpkg.com/heatmap.js"></script>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
     integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY="
     crossorigin=""/>
    <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
     integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo="
     crossorigin=""></script>
    <script src="lib/leaflet-heatmap.js"></script>


    <!-- Stylesheets -->
    <link rel="stylesheet" href="style.css">

    <!-- Main sketch file -->
    <script src="sketch.js"></script>

    <!-- Main visualisation files -->
    <script src="tech-diversity-race.js"></script>
    <script src="tech-diversity-gender.js"></script>
    <script src="pay-gap-by-job-2017.js"></script>
    <script src="pay-gap-1997-2017.js"></script>
    <script src="nutrients-1974-2016.js"></script>
    <script src="climate-change.js"></script>
    <script src="eating-habits.js"></script>
    <script src="air-quality.js"></script>
    <script src="seizures-2006-2018.js"></script>
    <script src="seizures-by-area.js"></script>

    <!-- Add extra scripts below -->
    <script src="helper-functions.js"></script>
    <script src="gallery.js"></script>
    <script src="pie-chart.js"></script>
    <script src="box.js"></script>
```

```
    <script src="waffle.js"></script>
    <script src="bar-chart.js"></script>
    <script src="heatmap.js"></script>

  </head>
  <body>
    <div id="app" class="container">
      <ul id="visuals-menu"></ul>
    </div>
  </body>
</html>
```

nutrients-1974-2016.js:

```
function NutrientsTimeSeries() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Nutrients: 1974-2016';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'nutrients-timeseries';

  // Title to display above the plot.
  this.title = 'Nutrient Consumption as a Percentage of Requirements 1974-2016';

  // Names for each axis.
  this.xAxisLabel = 'Year';
  this.yAxisLabel = '%';

  var marginSize = 35;

  this.colours = [];
  this.stats = [];

  // Layout object to store all common plot layout parameters and
  // methods.
  this.layout = {
    marginSize: marginSize,

    // Locations of margin positions. Left and bottom have double margin
    // size due to axis and tick labels.
    leftMargin: marginSize * 2,
    rightMargin: width - marginSize,
    topMargin: marginSize,
    bottomMargin: height - marginSize * 2,
    pad: 5,

    plotWidth: function() {
      return this.rightMargin - this.leftMargin;
    },

    plotHeight: function() {
      return this.bottomMargin - this.topMargin;
    },

    // Boolean to enable/disable background grid.
    grid: true,
```

```
    // Number of axis tick labels to draw so that they are not drawn on
    // top of one another.
    numXTickLabels: 10,
    numYTickLabels: 8,
  };

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
  this.preload = function() {
    var self = this;
    this.data = loadTable(
    './data/food/nutrients74-16.csv', 'csv', 'header',
    // Callback function to set the value
    // this.loaded to true.
    function(table) {
      self.loaded = true;
    });
  };


  this.setup = function() {
    // Font default
    textSize(16);

    // Set min and max years: assumes data is sorted by date.
    this.startYear = Number(this.data.columns[1]);
    this.endYear = Number(this.data.columns[this.data.columns.length-1]);

///// START OF MY CODE /////

    var minVal = Infinity;
    var maxVal = 0;

    // Store row and column count for reuse
    this.rowCount = this.data.getRowCount();
    this.colCount = this.data.getColumnCount();

    for(var i = 0; i < this.rowCount; i++){
      // Set random colour for each row of data
      this.colours.push(color(random(0,255),random(0,255),random(0,255)));

      // Find min for mapping to canvas height
      for(var j = 1; j < this.colCount; j++)
      {
```

```
            var val = this.data.getNum(i, j);
            if(val < minVal)
            {
               minVal = val;
            }
         }

         // Find max for mapping to canvas height
         for(var j = 1; j < this.colCount; j++)
         {
            var val = this.data.getNum(i, j);
            if(val > maxVal)
            {
               maxVal = val;
            }
         }
      }

      // Store these Min and Max values for drawing graph
      this.minPercentage = minVal;
      this.maxPercentage = maxVal;

      this.nutrients = [];

      // Get statistics for each row
      for (var i = 0; i < this.rowCount; i++) {
         var row = this.data.getRow(i);
         // Get nutrient name
         var l = row.getString(0);
         this.nutrients.push(l);
      }

      // Create a select DOM element.
      this.select = createSelect();
      this.select.position(1100,15);

      // Fill with options: First entry is empty.
      for (let i = 0; i < this.nutrients.length; i++)
      {
         this.select.option(this.nutrients[i]);
      }

      // Call to function that creates 2D array of stats
      this.getRowStats();
   };
```

```
  this.destroy = function() {
    this.select.remove();
  };
```

///// END OF MY CODE (CONTINUES BELOW)/////

```
  this.draw = function() {
    if (!this.loaded) {
      console.log('Data not yet loaded');
      return;
    }
    // Draw Title
    this.drawTitle();

    // Draw all y-axis labels.
    drawYAxisTickLabels(this.minPercentage,
                        this.maxPercentage,
                        this.layout,
                        this.mapPercentageToHeight.bind(this),
                        0);

    // Draw x and y axis.
    drawAxis(this.layout);

    // Draw x and y axis labels.
    drawAxisLabels(this.xAxisLabel,
                   this.yAxisLabel,
                   this.layout);

    // Plot all values between startYear and endYear using the width
    // of the canvas minus margins.
    var numYears = this.endYear - this.startYear;

    // Loop over all rows and draw a line from the previous value to
    // the current.
    for (var i = 0; i < this.rowCount; i++) {
      var row = this.data.getRow(i);
      var previous = null;

      var l = row.getString(0);

      for (var j = 1; j < numYears; j++)
      {
        // Create an object to store data for the current year.
        var current = {
          // Convert strings to numbers.
          'year': this.startYear + j - 1,
```

```
            'percentage': row.getNum(j)
          };

          if (previous != null) {
            // Draw line segment connecting previous year to current year.
            stroke(this.colours[i]);
            line(this.mapYearToWidth(previous.year),
                this.mapPercentageToHeight(previous.percentage),
                this.mapYearToWidth(current.year),
                this.mapPercentageToHeight(current.percentage));

            // The number of x-axis labels to skip so that only
            // numXTickLabels are drawn.
            var xLabelSkip = ceil(numYears / this.layout.numXTickLabels);

            // Draw the tick label marking the start of the previous year.///// MODIFIED THIS
CODE /////
            if (j % xLabelSkip == 0 && (i % 2 == 0)) {
              drawXAxisTickLabel(previous.year, this.layout,
                        this.mapYearToWidth.bind(this));
            }
          }
          else
          {
            noStroke();
            fill(this.colours[i]);
            text(l,105,this.mapPercentageToHeight(current.percentage));
          }
          // Assign current year to previous year so that it is available
          // during the next iteration of this loop to give us the start
          // position of the next line segment.
          previous = current;
        }
      }

      // Draw Stats box for the selected nutrient
      var nutrientName = this.select.value();
      this.drawStatsBox(nutrientName);
    };

    // Method to draw title
    this.drawTitle = function() {
      fill(0);
      noStroke();
      textAlign('center', 'center');

      text(this.title,
```

```
        (this.layout.plotWidth() / 2) + this.layout.leftMargin,
        this.layout.topMargin - (this.layout.marginSize / 2));
};

this.mapYearToWidth = function(value) {
  return map(value,
        this.startYear,
        this.endYear,
        this.layout.leftMargin,   // Draw left-to-right from margin.
        this.layout.rightMargin);
};

this.mapPercentageToHeight = function(value) {
  return map(value,
        this.minPercentage,
        this.maxPercentage,
        this.layout.bottomMargin, // Smaller value at bottom.
        this.layout.topMargin);   // Bigger value at top.
};
```

///// START OF MY CODE /////

```
// Create 2D array of Objects
this.getRowStats = function(){
  this.stats = [];
  for (var i = 0; i < this.rowCount; i++) {
    var row = this.data.getRow(i);
    // Get nutrient name
    var l = row.getString(0);
    //Create an array of row values to calculate stats
    // by passing these arrays to helper functions
    var rowArr = [];
    for (var j = 1; j < this.colCount; j++)
    {
      var val = row.getNum(j);
      rowArr.push(val);
    }
    //Creat an Array of objects for each row
    var rowStats = [];
    var rowAvg = {
      name: l,
      stat: "Average",
      value: getArrayAverage(rowArr)
      }
    var rowMed = {
      name: l,
      stat: "Median",
```

```
            value: getArrayMedian(rowArr)
            }
        var rowMax = {
            name: l,
            stat: "Max",
            value: max(rowArr)
            }
        var rowMin = {
            name: l,
            stat: "Min",
            value: min(rowArr)
            }
        rowStats.push(rowAvg);
        rowStats.push(rowMed);
        rowStats.push(rowMax);
        rowStats.push(rowMin);
        this.stats.push(rowStats);
    }
    return this.stats;
}

// Method to draw calculated statistics to the graph
this.drawStatsBox = function(nutrient) {
    var boxX = 800;
    var boxY = 40;
    push();
    noStroke();
    rect(boxX, boxY, 160, 100);
    pop();
    // Cycle through all stats for selected nutrient
    for(var i = 0; i < this.stats.length; i++)
    {
        // Cycle through sub array to access properties
        for(var j = 0; j < this.stats[i].length; j++){
            if(this.stats[i][j].name == nutrient){
                var boxY = 40 + j * 20;
                push();
                fill(255);
                noStroke();
                textAlign('left', 'center');
                textSize(14);
                text(this.stats[i][j].stat, boxX + 10, boxY + 20);
                text(this.stats[i][j].value  + " %", boxX + 80, boxY + 20);
                pop();
            }
        }
    }
```

```
    }

}

///// END OF MY CODE /////
```

pay-gap-1997-2017.js:

```
function PayGapTimeSeries() {

 // Name for the visualisation to appear in the menu bar.
 this.name = 'Pay gap: 1997-2017';

 // Each visualisation must have a unique ID with no special
 // characters.
 this.id = 'pay-gap-timeseries';

 // Title to display above the plot.
 this.title = 'Gender Pay Gap: Average difference between male and female pay.';

  // Names for each axis.
 this.xAxisLabel = 'year';
 this.yAxisLabel = '%';

 var marginSize = 35;

 // Layout object to store all common plot layout parameters and
 // methods.
 this.layout = {
  marginSize: marginSize,

  // Locations of margin positions. Left and bottom have double margin
  // size due to axis and tick labels.
  leftMargin: marginSize * 2,
  rightMargin: width - marginSize,
  topMargin: marginSize,
  bottomMargin: height - marginSize * 2,
  pad: 5,

  plotWidth: function() {
   return this.rightMargin - this.leftMargin;
  },

  plotHeight: function() {
   return this.bottomMargin - this.topMargin;
  },

  // Boolean to enable/disable background grid.
  grid: true,

  // Number of axis tick labels to draw so that they are not drawn on
  // top of one another.
  numXTickLabels: 10,
```

```
  numYTickLabels: 8,
};

// Property to represent whether data has been loaded.
this.loaded = false;

// Preload the data. This function is called automatically by the
// gallery when a visualisation is added.
this.preload = function() {
  var self = this;
  this.data = loadTable(
    './data/pay-gap/all-employees-hourly-pay-by-gender-1997-2017.csv', 'csv', 'header',
    // Callback function to set the value
    // this.loaded to true.
    function(table) {
      self.loaded = true;
    });

};

this.setup = function() {
  // Font defaults.
  textSize(16);

  // Set min and max years: assumes data is sorted by date.
  this.startYear = this.data.getNum(0, 'year');
  this.endYear = this.data.getNum(this.data.getRowCount() - 1, 'year');

  // Find min and max pay gap for mapping to canvas height.
  this.minPayGap = 0;         // Pay equality (zero pay gap).
  this.maxPayGap = max(this.data.getColumn('pay_gap'));
};

this.destroy = function() {
};

this.draw = function() {
  if (!this.loaded) {
    console.log('Data not yet loaded');
    return;
  }

  // Draw the title above the plot.
  this.drawTitle();

  // Draw all y-axis labels.
  drawYAxisTickLabels(this.minPayGap,
```

```
            this.maxPayGap,
            this.layout,
            this.mapPayGapToHeight.bind(this),
            0);

// Draw x and y axis.
drawAxis(this.layout);

// Draw x and y axis labels.
drawAxisLabels(this.xAxisLabel,
          this.yAxisLabel,
          this.layout);

// Plot all pay gaps between startYear and endYear using the width
// of the canvas minus margins.
var previous;
var numYears = this.endYear - this.startYear;

// Loop over all rows and draw a line from the previous value to
// the current.
for (var i = 0; i < this.data.getRowCount(); i++) {

 // Create an object to store data for the current year.
 var current = {
  // Convert strings to numbers.
  'year': this.data.getNum(i, 'year'),
  'payGap': this.data.getNum(i, 'pay_gap')
 };

 if (previous != null) {
  // Draw line segment connecting previous year to current
  // year pay gap.
  stroke(0);
  line(this.mapYearToWidth(previous.year),
      this.mapPayGapToHeight(previous.payGap),
      this.mapYearToWidth(current.year),
      this.mapPayGapToHeight(current.payGap));

  // The number of x-axis labels to skip so that only
  // numXTickLabels are drawn.
  var xLabelSkip = ceil(numYears / this.layout.numXTickLabels);

  // Draw the tick label marking the start of the previous year.
  if (i % xLabelSkip == 0) {
   drawXAxisTickLabel(previous.year, this.layout,
              this.mapYearToWidth.bind(this));
  }
```

```
    }

    // Assign current year to previous year so that it is available
    // during the next iteration of this loop to give us the start
    // position of the next line segment.
    previous = current;
  }
 };

  // Method to draw title
  this.drawTitle = function() {
     push();
     fill(0);
     textSize(22);
     noStroke();
     textAlign('center', 'center');

     text(this.title,
        (this.layout.plotWidth() / 2) + this.layout.leftMargin,
        this.layout.topMargin - (this.layout.marginSize / 2));
     pop();
  };

 this.mapYearToWidth = function(value) {
  return map(value,
          this.startYear,
          this.endYear,
          this.layout.leftMargin,   // Draw left-to-right from margin.
          this.layout.rightMargin);
 };

 this.mapPayGapToHeight = function(value) {
  return map(value,
          this.minPayGap,
          this.maxPayGap,
          this.layout.bottomMargin, // Smaller pay gap at bottom.
          this.layout.topMargin);   // Bigger pay gap at top.
 };
}
```

pay-gap-by-job-2017.js:

```
function PayGapByJob2017() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Pay gap by Occupation: 2017';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'pay-gap-by-job-2017';

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Graph properties.
  this.pad = 50;
  this.dotSizeMin = 15;
  this.dotSizeMax = 40;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
  this.preload = function() {
    var self = this;
    this.data = loadTable(
    './data/pay-gap/occupation-hourly-pay-by-gender-2017.csv', 'csv', 'header',
    // Callback function to set the value
    // this.loaded to true.
    function(table) {
    self.loaded = true;
    });

  };

///// START OF MY CODE /////

  this.setup = function() {

    // (Adapted from original code)
    // Get necessary data from the table object.
    this.jobs = this.data.getColumn('job_subtype');
    this.propFemale = this.data.getColumn('proportion_female');
    this.payGap = this.data.getColumn('pay_gap');
    this.numJobs = this.data.getColumn('num_jobs');

    // (Adapted from original code)
    // Convert numerical data from strings to numbers.
    this.propFemale = stringsToNumbers(this.propFemale);
```

```
    this.payGap = stringsToNumbers(this.payGap);
    this.numJobs = stringsToNumbers(this.numJobs);

    // (Adapted from original code)
    // Set ranges for axes.
    // Use full 100% for x-axis (proportion of women in roles).
    var propFemaleMin = 0;
    var propFemaleMax = 100;

    // (Adapted from original code)
    // For y-axis (pay gap) use a symmetrical axis equal to the
    // largest gap direction so that equal pay (0% pay gap) is in the
    // centre of the canvas. Above the line means men are paid
    // more. Below the line means women are paid more.
    var payGapMin = -20;
    var payGapMax = 20;

    // Find smallest and largest numbers of people across all
    // categories to scale the size of the dots.
    var numJobsMin = min(this.numJobs);
    var numJobsMax = max(this.numJobs);

    // Create array of ellipses to store objects for each data point
    this.ellipses = [];

    for (i = 0; i < this.data.getRowCount(); i++) {
      // map X, Y and Size to proper proportions
      var ellipseX = map(this.propFemale[i], propFemaleMin, propFemaleMax, this.pad,
width - this.pad);
      var ellipseY = map(this.payGap[i], payGapMin, payGapMax, height - this.pad,
this.pad);
      var ellipseSize = map(this.numJobs[i], numJobsMin, numJobsMax,this.dotSizeMin,
this.dotSizeMax);

      // Use conditionals to define colour and store in data point object
      var ellipseColour;
      // Get colour based on pay gap size
      if (abs(this.payGap[i]) < 5) {
        ellipseColour = "Green"; // Green for less than 5 Percent
      } else if (abs(this.payGap[i]) < 10) {
        ellipseColour = "Yellow"; // Yellow for less than 10 percent
      } else {
        ellipseColour = "Red"; // Red for greater than 10%
      }
      this.ellipses.push(
        {
          x: ellipseX,
```

```
            y: ellipseY,
            size: ellipseSize,
            colour: ellipseColour
          }
        );
      }
    };
```

///// END OF MY CODE (Continues below)/////

///// MODIFIED CODE BELOW /////

```
    this.draw = function() {
      if (!this.loaded) {
        console.log('Data not yet loaded');
        return;
      }

      // Draw the axes.
      this.addAxes();
```

///// START OF MY CODE /////

```
      // Draw title
      this.drawTitle();

      stroke(0);
      strokeWeight(1);

      // Draw ellipses using object properties
      for (i = 0; i < this.ellipses.length; i++) {
        // Draw an ellipse for each point.
        // x = propFemale
        // y = payGap
        // size = numJobs
        fill(this.ellipses[i].colour);
        ellipse(this.ellipses[i].x, this.ellipses[i].y, this.ellipses[i].size);
      }

      // Print data when hovering over ellipse
      for (i = 0; i < this.ellipses.length; i++) {
        // Check if mouseOver using method below
        var mouseOver = this.mouseOver(mouseX, mouseY, this.ellipses[i].x, this.ellipses[i].y,
this.ellipses[i].size);
        if(mouseOver != false){
          // Draw Name of Job, Prop. Female, and Pay gap when hovering
```

```
            push();
            fill(0);
            textSize(15);
            var tWidth = textWidth(this.jobs[i]);
            textAlign(LEFT, TOP);
            rect(mouseX - 80, mouseY, tWidth + 30, 75);
            fill(255);
            text(this.jobs[i], mouseX - 70, mouseY + 10);
            text("Pay Gap: " + this.payGap[i].toFixed(2) + " %", mouseX - 70, mouseY + 30);
            text("Prop. Female: " + this.propFemale[i].toFixed(2) + " %", mouseX - 70, mouseY +
50);
            pop();
            break;
          }
        }
    };

///// END OF MY CODE (CONTINUES BELOW)/////

    this.addAxes = function () {
        stroke(200);

        // Add vertical line.
        line(width / 2,
            0 + this.pad,
            width / 2,
            height - this.pad);

        // Add horizontal line.
        line(0 + this.pad,
            height / 2,
            width - this.pad,
            height / 2);

///// START OF MY CODE /////

        // Add Axis titles and subtitles
        push();
        fill(0);
        textSize(13);
        textAlign(LEFT);
        text("Wage Disparity", this.pad, height/2 - 8);
        rotate(-PI/2);
        text("Proportion Male to Female", -height + this.pad, width/2 - 8);

        // Subtitles
        textAlign(CENTER,CENTER);
```

```
      fill(150);
      textSize(20);

      // Y Axis subtitle
      text("Higher Wages for Males", -160, width - this.pad);
      text("Higher Wages for Females", -430, width - this.pad);
      rotate(PI/2);

      // X Axis subtitle
      text("Male Dominated", width/4, height - 12);
      text("Female Dominated", width*3/4, height - 12);
      pop();
   };

   // Method to draw title
   this.drawTitle = function() {
      fill(0);
      noStroke();
      textSize(22);
      textAlign(CENTER,CENTER);
      text(this.name,width/2, 20);
   };

   // Method to check mouse over
   this.mouseOver = function(mouseX, mouseY, x, y, size){
      if(mouseX > x - size/2 && mouseX < x + size/2 && mouseY > y - size/2 && mouseY < y +
size/2)
      {
         return true;
      }
      return false;
   }
};

///// END OF MY CODE /////
```

pie-chart.js:

```
function PieChart(x, y, diameter) {

  this.x = x;
  this.y = y;
  this.diameter = diameter;
  this.labelSpace = 25;

  this.get_radians = function(data) {
    var total = sum(data);
    var radians = [];

    for (let i = 0; i < data.length; i++) {
      radians.push((data[i] / total) * TWO_PI);
    }

    return radians;
  };

  this.draw = function(data, labels, colours, title) {

    // Test that data is not empty and that each input array is the
    // same length.
    if (data.length == 0) {
      alert('Data has length zero!');
    } else if (![labels, colours].every((array) => {
      return array.length == data.length;
    })) {
      alert(`Data (length: ${data.length})
Labels (length: ${labels.length})
Colours (length: ${colours.length})
Arrays must be the same length!`);
    }

    // https://p5js.org/examples/form-pie-chart.html

    var angles = this.get_radians(data);
    var lastAngle = 0;
    var colour;

    for (var i = 0; i < data.length; i++) {
      if (colours) {
        colour = colours[i];
      } else {
        colour = map(i, 0, data.length, 0, 255);
      }
```

```
    fill(colour);
    stroke(0);
    strokeWeight(1);

    arc(this.x, this.y,
       this.diameter, this.diameter,
       lastAngle, lastAngle + angles[i] + 0.001); // Hack for 0!

    if (labels) {
     this.makeLegendItem(labels[i], i, colour);
    }

    lastAngle += angles[i];
   }

   if (title) {
    noStroke();
    textAlign('center', 'center');
    textSize(22);
    text(title, this.x, this.y - this.diameter * 0.6);
   }
  };

  this.makeLegendItem = function(label, i, colour) {
   var x = this.x + 50 + this.diameter / 2;
   var y = this.y + (this.labelSpace * i) - this.diameter / 2;
   var boxWidth = this.labelSpace / 2;
   var boxHeight = this.labelSpace / 2;

   fill(colour);
   rect(x, y, boxWidth, boxHeight);

   fill('black');
   noStroke();
   textAlign('left', 'center');
   textSize(12);
   text(label, x + boxWidth + 10, y + boxWidth / 2);
  };
}
```

seizures-2006-2018.js:

///// CODE ADAPATED AND MODED FROM TECH DIVERSITY RACE CODE/////

```
function NumberOfSeizures() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Seizures by Substance: 2006-2018';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'seizures-by-substance';

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
  this.preload = function() {
    var self = this;
    this.data = loadTable(
      './data/seizures/seizures-uk-06-18.csv', 'csv', 'header',
      // Callback function to set the value
      // this.loaded to true.
      function(table) {
        self.loaded = true;
      });
  };

  this.setup = function() {
    if (!this.loaded) {
      console.log('Data not yet loaded');
      return;
    }

    // Create a select DOM element.
    this.select = createSelect();
    this.select.position(350, 40);

    // Fill the options with all substance names.
    var years = this.data.columns;
    // First entry is empty.
    for (var i = 1; i < years.length; i++) {
      this.select.option(years[i]);
    }

    this.colours = [];
```

```
  for(var i = 0; i < this.data.getRowCount(); i++){
     // Colour to use for each category.
     this.colours.push(color(random(0,255),random(0,255),random(0,255)));
  }
 };

 this.destroy = function() {
  this.select.remove();
 };

 // Create a new pie chart object.
 this.pie = new PieChart(width / 2, height / 2, width * 0.4);

 this.draw = function() {
  if (!this.loaded) {
   console.log('Data not yet loaded');
   return;
  }

  // Get the value of the company we're interested in from the
  // select item.
  var selectedYear = this.select.value();

  // Get the column of raw data for years.
  var col = this.data.getColumn(selectedYear);

  // Convert all data strings to numbers.
  col = stringsToNumbers(col);

  // Copy the row labels from the table (the first item of each row).
  var labels = this.data.getColumn(0);

  // Make a title.
  var title = "Seizures by Substance: " + selectedYear;

  // Draw the pie chart!
  this.pie.draw(col, labels, this.colours, title);
 };
}
```

seizures-by-area.js:

```
function SeizuresByArea() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'UK Seizures by Region';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'seizures-by-region';

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
  this.preload = function() {
    var self = this;
    this.data = loadTable(
      './data/seizures/seizures-by-area-06-18.csv', 'csv', 'header',
      // Callback function to set the value
      // this.loaded to true.
      function(table) {
        self.loaded = true;
    });
  };

///// START OF MY CODE /////

  this.setup = function() {
    if (!this.loaded) {
      console.log('Data not yet loaded');
      return;
    }

    // Create a select DOM element.
    this.select = createSelect();
    this.select.position(400, 20);

    // Fill the options with all years names.
    var years = this.data.columns;
    // First entry is empty.
    for (var i = 3; i < years.length; i++) {
      this.select.option(years[i]);
    }

    // Create new base map layer with coordinates of UK and zoom level
```

```
    this.mapLayer = new HeatMap(54.7023545, -3.2765753, 5);

    };

  // Remove visual (select object and map object)
  this.destroy = function() {
    this.select.remove();
    this.mapLayer.mapDestroy();

  };

  this.draw = function() {
    if (!this.loaded) {
      console.log('Data not yet loaded');
    return;
    }

    // Get the value of the selected year for title
    var selectedYear = this.select.value();
    this.drawTitle(selectedYear);


  };

  // Method to draw title
  this.drawTitle = function(value) {
    fill(0);
    noStroke();
    textSize(22);
    textAlign(CENTER,CENTER);
    text(this.name + ": " + value, width/2, 20);
  };
}

///// END OF MY CODE /////
```

sketch.js:

```
// Global variable to store the gallery object. The gallery object is
// a container for all the visualisations.
var gallery;

function setup() {
   // Create a canvas to fill the content div from index.html.
   canvasContainer = select('#app');
   var c = createCanvas(1024, 576);
   c.parent('app');

   // Create a new gallery object.
   gallery = new Gallery();

   // Add the visualisation objects here.
   gallery.addVisual(new TechDiversityRace());
   gallery.addVisual(new TechDiversityGender());
   gallery.addVisual(new PayGapByJob2017());
   gallery.addVisual(new PayGapTimeSeries());
   gallery.addVisual(new ClimateChange());
   gallery.addVisual(new NutrientsTimeSeries());
   gallery.addVisual(new EatingHabits());
   gallery.addVisual(new AirQuality());
   gallery.addVisual(new NumberOfSeizures());
   gallery.addVisual(new SeizuresByArea());

}

function draw() {
  background(255);
  if (gallery.selectedVisual != null) {
    gallery.selectedVisual.draw();
  }
}
```

style.css:

```
body{
   font-family: sans-serif
}

.menu-item {
   width: 250px;
   height: 40px;
   border: 1px solid gray;
   list-style-type: none;
   padding: 5px;
}

.container {
  display: flex;
}

.hover {
   background: lightgray
}

.selected{
   background: gold
}


/* START OF MY CODE */
#map {
   height: 450px;
   width: 850px;
}
```

tech-diversity-gender.js:

```
function TechDiversityGender() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Tech Diversity: Gender';

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'tech-diversity-gender';

  // Layout object to store all common plot layout parameters and
  // methods.
  this.layout = {
    // Locations of margin positions. Left and bottom have double margin
    // size due to axis and tick labels.
    leftMargin: 130,
    rightMargin: width,
    topMargin: 80,
    bottomMargin: height,
    pad: 5,

    plotWidth: function() {
      return this.rightMargin - this.leftMargin;
    },

    // Boolean to enable/disable background grid.
    grid: true,

    // Number of axis tick labels to draw so that they are not drawn on
    // top of one another.
    numXTickLabels: 10,
    numYTickLabels: 8,
  };

  // Middle of the plot: for 50% line.
  this.midX = (this.layout.plotWidth() / 2) + this.layout.leftMargin;

  // Default visualisation colours.
  this.femaleColour = color('#FFCCFF');
  this.maleColour = color('#66B2FF');

  // Property to represent whether data has been loaded.
  this.loaded = false;

  // Preload the data. This function is called automatically by the
  // gallery when a visualisation is added.
```

```
this.preload = function() {
  var self = this;
  this.data = loadTable(
  './data/tech-diversity/gender-2018.csv', 'csv', 'header',
  // Callback function to set the value
  // this.loaded to true.
  function(table) {
    self.loaded = true;
  });

};

this.setup = function() {
  // Font defaults.
  textSize(16);
};

this.destroy = function() {
};

this.draw = function() {
  if (!this.loaded) {
    console.log('Data not yet loaded');
    return;
  }

// Draw title
this.drawTitle();

// Draw Female/Male labels at the top of the plot.
this.drawCategoryLabels();

var lineHeight = (height - this.layout.topMargin) /
  this.data.getRowCount();

for (var i = 0; i < this.data.getRowCount(); i++) {

 // Calculate the y position for each company.
 var lineY = (lineHeight * i) + this.layout.topMargin;

 // Create an object that stores data from the current row.
 var company = {
  // Convert strings to numbers.
  'name': this.data.getString(i, 'company'),
  'female': this.data.getNum(i, 'female'),
  'male': this.data.getNum(i, 'male'),
 };
```

```
  // Draw the company name in the left margin.
  fill(0);
  noStroke();
  textAlign('right', 'top');
  text(company.name,
      this.layout.leftMargin - this.layout.pad,
      lineY);

  // Draw female employees rectangle.
  fill(this.femaleColour);
  rect(this.layout.leftMargin,
      lineY,
      this.mapPercentToWidth(company.female),
      lineHeight - this.layout.pad);

  // Draw male employees rectangle.
  fill(this.maleColour);
  rect(this.layout.leftMargin + this.mapPercentToWidth(company.female),
      lineY,
      this.mapPercentToWidth(company.male),
      lineHeight - this.layout.pad);
}

// Draw 50% line
stroke(150);
strokeWeight(1);
line(this.midX,
    this.layout.topMargin,
    this.midX,
    this.layout.bottomMargin);

};

this.drawCategoryLabels = function() {
   fill(0);
   noStroke();
   textAlign('left', 'top');
   text('Female',
       this.layout.leftMargin, this.layout.topMargin - 20);
   textAlign('center', 'top');
   text('50%',
       this.midX,
       this.layout.topMargin - 20);
   textAlign('right', 'top');
   text('Male',
       this.layout.rightMargin,
```

```
        this.layout.topMargin - 20);
  };

  this.mapPercentToWidth = function(percent) {
    return map(percent,
          0,
          100,
          0,
          this.layout.plotWidth());
  };

  this.drawTitle = function() {
    push();
    fill(0);
    noStroke();
    textSize(24);
    textAlign(CENTER,CENTER);
    text(this.name,width/2 + 2 * this.layout.pad, 20);
    pop();
  };
}
```

tech-diversity-race.js:

```
function TechDiversityRace() {

 // Name for the visualisation to appear in the menu bar.
 this.name = 'Tech Diversity: Race';

 // Each visualisation must have a unique ID with no special
 // characters.
 this.id = 'tech-diversity-race';

 // Property to represent whether data has been loaded.
 this.loaded = false;

 // Preload the data. This function is called automatically by the
 // gallery when a visualisation is added.
 this.preload = function() {
  var self = this;
  this.data = loadTable(
   './data/tech-diversity/race-2018.csv', 'csv', 'header',
   // Callback function to set the value
   // this.loaded to true.
   function(table) {
    self.loaded = true;
   });
 };

 this.setup = function() {
  if (!this.loaded) {
   console.log('Data not yet loaded');
   return;
  }

  // Create a select DOM element.
  this.select = createSelect();
  this.select.position(350, 40);

  // Fill the options with all company names.
  var companies = this.data.columns;
  // First entry is empty.
  for (var i = 1; i < companies.length; i++) {
   this.select.option(companies[i]);
  }
 };

 this.destroy = function() {
  this.select.remove();
```

```
  };

  // Create a new pie chart object.
  this.pie = new PieChart(width / 2, height / 2, width * 0.4);

  this.draw = function() {
    if (!this.loaded) {
      console.log('Data not yet loaded');
      return;
    }

    // Get the value of the company we're interested in from the
    // select item.
    var companyName = this.select.value();

    // Get the column of raw data for companyName.
    var col = this.data.getColumn(companyName);

    // Convert all data strings to numbers.
    col = stringsToNumbers(col);

    // Copy the row labels from the table (the first item of each row).
    var labels = this.data.getColumn(0);

    // Colour to use for each category.
    var colours = ['blue', 'red', 'green', 'pink', 'purple', 'yellow'];

    // Make a title.
    var title = 'Employee diversity at ' + companyName;

    // Draw the pie chart!
    this.pie.draw(col, labels, colours, title);
  };
}
```

waffle.js:

```
function Waffle(x, y, width, height, boxes_across, boxes_down, table, columnHeading, possibleValues, title){

    var x = x;
    var y = y;
    var width = width;
    var height = height;
    var boxes_across = boxes_across;
    var boxes_down = boxes_down;
    var title = title;

    // Determine for which column to create waffle
    var column = table.getColumn(columnHeading);
    var possibleValues = possibleValues;

    var colours = ["red", "green", "blue", "purple", "yellow", "orange"];

    var categories = [];
    var boxes = [];

    function categoryLocation(categoryName){
        for (var i = 0; i < categories.length; i++){
            if(categoryName == categories[i].name){
                return i;
            }
        }
        return -1;  // Handle the error**
    }

    function addCategories(){
        for(var i = 0; i < possibleValues.length; i++){
            categories.push({
                "name": possibleValues[i],
                "count": 0,
                "colour": colours[i % colours.length]
            })
        }

        for (var i = 0; i < column.length; i++){
            var catLocation = categoryLocation(column[i]);

            if(catLocation != -1){         // Error handling **
                categories[catLocation].count++;
            }
        }
```

```
    // Iterate over categories and add proportions
    for(var i = 0; i < categories.length; i++){
      categories[i].boxes = round((categories[i].count / column.length) * (boxes_down *
boxes_across));
    }
  }

  function addBoxes(){
    var currentCategory = 0;
    var currentCategoryBox = 0;

    var boxWidth = width / boxes_across;
    var boxHeight = height / boxes_down;

    for(var i = 0; i < boxes_down; i++){
      boxes.push([]);
      for(var j = 0; j < boxes_across; j++){
        if(currentCategoryBox == categories[currentCategory].boxes){
          currentCategoryBox = 0;
          currentCategory++;
        }

        boxes[i].push(new Box((x + (j*boxWidth)), (y + (i*boxHeight)), boxWidth,
boxHeight, categories[currentCategory]));

        currentCategoryBox++;
      }
    }
  }

  addCategories();
  addBoxes();

  this.draw = function(){
    // zdraw waffle diagram
    stroke(0);
    for(var i = 0; i < boxes.length; i++){
      for(var j = 0; j < boxes[i].length; j++){
        if(boxes[i][j].category != undefined){
          boxes[i][j].draw();
        }
      }
    }
    push();
    noStroke();
    fill(100);
```

```
    textSize(18);
    textAlign(CENTER,CENTER);
    text(title, x + width/2, y - 20);
    pop();

  }

  // Method to draw name of category when hovered over boxes
  this.checkMouse = function(mouseX, mouseY){
    for(var i = 0; i < boxes.length; i++){
      for(var j = 0; j < boxes[i].length; j++){
        if(boxes[i][j].category != undefined){
        var mouseOver = boxes[i][j].mouseOver(mouseX, mouseY);
        if(mouseOver != false){
          push();
          fill(0);
          textSize(20);
          var tWidth = textWidth(mouseOver);
          textAlign(LEFT, TOP);
          rect(mouseX, mouseY, tWidth + 20, 40);
          fill(255);
          text(mouseOver, mouseX + 10, mouseY + 10);
          pop();
          break;
        }
        }
      }
    }
  }
}
```