1.surfaceTemperature.js

function SurfaceTemperature() {

  // Name for the visualisation to appear in the menu bar.

  this.name = 'Surface Temperature: 1880-2018';

  this.id = 'surface-temperature-timeseries';

  // Title to display above the plot.

  this.title = 'Land-Ocean: Surface Temperature Means(From 1880 to 2018)';

    // Names for each axis.

  this.xAxisLabel = 'year';

  this.yAxisLabel = 'Temperature';

  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: 1,

    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);

    // 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');

//original part begin

    // Find min and max temperatures for mapping to canvas height. Rather than use the code defines,

    // Travesing the data to find the exact ones.

    this.minTemperature = this.data.getNum(0, 'temperature');

    for(i = 1; i < this.data.getRowCount(); i++){

      if(this.data.getNum(i, 'temperature') < this.minTemperature){

        this.minTemperature = this.data.getNum(i, 'temperature');

      }

    }

    this.maxTemperature = this.data.getNum(0, 'temperature');

    for(i = 1; i < this.data.getRowCount(); i++){

      if(this.data.getNum(i, 'temperature') > this.maxTemperature){

        this.maxTemperature = this.data.getNum(i, 'temperature');

      }

    }

  };

//original part end

  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.minTemperature,

                        this.maxTemperature,

                        this.layout,

                        this.mapTemperatureToHeight.bind(this),

                        3);

    // Draw x and y axis.

    drawAxis(this.layout);

    // Draw x and y axis labels.

    drawAxisLabels(this.xAxisLabel,

                   this.yAxisLabel,

                   this.layout);

    // Plot all temperaturess 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 = {

        'year': this.data.getNum(i, 'year'),

        'temperature': this.data.getNum(i, 'temperature')

      };

      if (previous != null) {

        // Draw line segment connecting previous year to current

        // year temperatures.

        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);

        var xLabelSkip = 10;

        // 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;

    }

  };

  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.mapTemperatureToHeight = function(value) {

    return map(value,

               this.minTemperature,

               this.maxTemperature,

               this.layout.bottomMargin, // Smaller temperatures at bottom.

               this.layout.topMargin);   // Bigger temperatures at top.

  };

}

2.personal\_incomes\_2010\_2021.js

function PersonalIncomes() {

//original part begin//

  // Name for the visualisation to appear in the menu bar.

  this.name = 'Personal Incomes After Tax: 2010-20';

  // Each visualisation must have a unique ID with no special

  // characters.

  this.id = 'personal-incomes-after-tax-2010-2021';

  // Title to display above the plot.

  this.title = 'Personal Incomes After Tax in the UK: 2010-20';

    // Names for each axis.

  this.xAxisLabel = 'year';

  this.yAxisLabel = 'Personal Incomes After Tax';

//original part end

this.colors = [];

  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 - 8,

    rightMargin: width - marginSize,

    topMargin: marginSize,

bottomMargin: height - marginSize \* 2,

//original part begin

pad: 5,

//original part end

    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.

//original part begin

    numXTickLabels: 10,

numYTickLabels: 8,

//original part end

  };

  // 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/personal-Incomes/PersonalIncomesAfterTax(2010-2021).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);

//original part begin

    // Get min and max years:

    this.endYear = 2021;

    this.startYear = 2010;

    this.minY = 0;

    this.maxY = 130000;

this.series = {};

//original part end

    //loop over all the rows

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

    {

        var row = this.data.getRow(i);

        //if the series isn't there already add a new array

        if(this.series[row.getString(0)] == undefined)

        {

            this.series[row.getString(0)] = [];

            this.colors.push(color(random(0,255),random(0,255),random(0,255)));

        }

        for(var j = 1; j < this.data.getColumnCount(); j++)

        {

            this.minY =  min(this.minY, row.getNum(j));

            this.maxY = max(this.maxY, row.getNum(j));

            // we are assuming that the data is in chronological order

            this.series[row.getString(0)].push(row.getNum(j));

        }

    }

  };

  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.minY,

                        this.maxY,

                        this.layout,

                        this.mapYToHeight.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 incomess between startYear and endYear using the width

    // of the canvas minus margins.

    var numYears = this.endYear - this.startYear;

    for(var i = 0; i < numYears; i++)

    {

        // The number of x-axis labels to skip so that only

        // numXTickLabels are drawn.

        var xLabelSkip = ceil(numYears / this.layout.numXTickLabels);

        y = this.startYear + i;

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

        if (i % xLabelSkip == 0) {

          drawXAxisTickLabel(y, this.layout,

                             this.mapYearToWidth.bind(this));

        }

    }

    var legend = Object.keys(this.series);

    for(var j = 0; j < legend.length; j++)

    {

        var previous = null;

        // Loop over all rows and draw a line from the previous value to

        // the current.

        for (var i = 0; i < this.series[legend[j]].length; i++)

        {

          // Create an object to store data for the current year.

          var current = {

            // Convert strings to numbers.

            'year': this.startYear + i,

            'personalIncomes': this.series[legend[j]][i]

          };

          if (previous != null) {

            // Draw line segment connecting previous year to current

            // year incomes.

            stroke(this.colors[j]);

            line(this.mapYearToWidth(previous.year),

                 this.mapYToHeight(previous.personalIncomes),

                 this.mapYearToWidth(current.year),

                 this.mapYToHeight(current.personalIncomes));

          }

          else

          {

            push();

            textAlign(LEFT);

            noStroke();

            // adjust the y coordinates to look better.

            text(legend[j], 100 ,this.mapYToHeight(current.personalIncomes) - 5);

            pop();

          }

          // 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;

        }

    }

  };

  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.mapYToHeight = function(value) {

    return map(value,

               this.minY,

               this.maxY,

               this.layout.bottomMargin, // Smaller incomes at bottom.

               this.layout.topMargin);   // Bigger incomes at top.

  };

}

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

function PayGapByJob2017() {

  // Name for the visualisation to appear in the menu bar.

  //original part begin

this.name = 'Pay gap by job: 2017';

  // Each visualisation must have a unique ID with no special

  // characters.

  this.id = 'pay-gap-by-job-2017';

//original part end

  // Property to represent whether data has been loaded.

  this.loaded = false;

  // Graph properties.

  this.pad = 20;

  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;

      });

  };

  this.setup = function() {

  };

  this.destroy = function() {

  };

  this.draw = function() {

    if (!this.loaded) {

      console.log('Data not yet loaded');

      return;

    }

    // Draw the axes.

    this.addAxes();

    // Get data from the table object.

    var jobs = this.data.getColumn('job\_subtype');

    var propFemale = this.data.getColumn('proportion\_female');

    var payGap = this.data.getColumn('pay\_gap');

    var numJobs = this.data.getColumn('num\_jobs');

    // Convert numerical data from strings to numbers.

    propFemale = stringsToNumbers(propFemale);

    payGap = stringsToNumbers(payGap);

    numJobs = stringsToNumbers(numJobs);

    // Set ranges for axes.

    //

    // Use full 100% for x-axis (proportion of women in roles).

    var propFemaleMin = 0;

    var propFemaleMax = 100;

    // 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(numJobs);

    var numJobsMax = max(numJobs);

    stroke(0);

    strokeWeight(1);

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

      // Draw an ellipse for each point.

      // x = propFemale

      // y = payGap

      // size = numJobs

//original part begin

      var numJobscol = map(numJobs[i], numJobsMin, numJobsMax, 0, 255);

      console.log(numJobscol/255);

      fill(colour(numJobscol, 255 - numJobscol, 0, numJobscol/255), numJobscol/255);

//original part end

      ellipse(

        map(propFemale[i], propFemaleMin, propFemaleMax,

            this.pad, width - this.pad),

        map(payGap[i], payGapMin, payGapMax,

            height - this.pad, this.pad),

        map(numJobs[i], numJobsMin, numJobsMax,

            this.dotSizeMin, this.dotSizeMax)

      );

    }

  };

  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);

  };

}