```
// 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 Waffles());
    gallery.addVisual(new foodDataUK());
    }

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

```
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();
        }
    };
}
```

```
// ---------------------------------------------------------------------
// 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 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;
    }
};

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.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.
};
}
```

```
function PayGapByJob2017() {

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

    // 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.
        // Data for the chart
        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');

        // Data for info box
        var jobType = this.data.getColumn('job_type');
        var jobSubType = this.data.getColumn('job_subtype');
        var numJobsMale = this.data.getColumn('num_jobs_male');
        var medianMale = this.data.getColumn('median_male');
        var numJobsFemale = this.data.getColumn('num_jobs_female');
```

```
        var medianFemale = this.data.getColumn('median_female');

        // 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++) {
            //Temperature based on numJobs, to change in the future
            fill(map(numJobs[i], numJobsMin, numJobsMax, 0, 255));
            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);

        // Add labels for axes
        textAlign(CENTER, TOP);
        textSize(14);
        fill(0);
```

```
        text('proportion of Women in Roles (%)', width/2, height - this.pad + 5);

        textAlign(CENTER, LEFT);
        text('Pay Gap', this.pad + 20, height/2);
    };
}
```

```
function ClimateChange() {

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

    // 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;
    }

    // Calculate the average temperature change for the displayed data range.
    var startYearIndex = this.startYear - this.minYear;
    var endYearIndex = this.endYear - this.minYear;

    var startTemperature = this.data.getColumn('temperature')[startYearIndex];
    var endTemperature = this.data.getColumn('temperature')[endYearIndex];

    var averageTemperatureChange = endTemperature - startTemperature;
```

```
// 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 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++;

        // Draw the average temperature change text box.
        fill(0);
        textSize(20);
        textAlign('center');
        text(
            'Average Temperature Change: ' +
averageTemperatureChange.toFixed(2) + '℃',
            width / 2,
```

```
                this.layout.topMargin + 60
            );
        };

        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);
        };
}
```

```
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 (let 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);
    };
}
```

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

    this.x = x;
    this.y = y;
    this.diameter = diameter;
    this.labelSpace = 30;

    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, total) {

        // 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) {
                // Calculate the total value of data
                var totalValue = data.reduce((acc, value) => acc + value, 0);
```

```
                // Calculate percentage and display it in the legend
                var percentage = ((data[i] / totalValue) * 100).toFixed(2);
                this.makeLegendItem(labels[i] + ' (' + percentage + '%)', i,
colour);
            }



            lastAngle += angles[i];
            }

            if (title) {
                noStroke();
                textAlign('center', 'center');
                textSize(24);
                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 / 3;
        var boxWidth = this.labelSpace / 2;
        var boxHeight = this.labelSpace / 2;

        // Adjust the position of the text
        var textX = x + boxWidth + 20;
        var textY = y + boxHeight / 2 + 3;

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

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

```
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: 30,
        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(255, 0 ,0);
    this.maleColour = color(0, 255, 0);

    // 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 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.pad);
    textAlign('center', 'top');
    text('50%',
        this.midX,
        this.layout.pad);
    textAlign('right', 'top');
    text('Male',
        this.layout.rightMargin,
        this.layout.pad);
};

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

```
function Waffles(){

    //properties
    this.name = 'How Do People Prepare Meals';
    this.id = 'how-do-people-prepare-meals';
    this.title = 'How Do People Prepare Meals, Generated Survey';
    this.loaded = false;
    var data;
    var waffles = [];

    //preload data
    this.preload = function(){
        var self = this;
        data = loadTable("./data/uk-food/meals.csv", "csv", "header",
        function(table) {
            self.loaded = true;
        });
    }

    //Setup waffles
    this.setup = function() {
        var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"]
        var values = ['Take-away', 'Cooked from fresh', 'Ready meal', 'Ate out',
'Skipped meal', 'Left overs']

        //box colours
        var colours = [];
        //random colours generator
        for (let i = 0; i < values.length; i++) {
            //i is included to ensure no 2 colours are the same
            let r = random(0 + i * 20, 255);  // Red
            let g = random(0 + i * 20, 255);  // Green
            let b = random(0 + i * 20, 255);  // Blue
            colours.push(color(r, g, b));
        }

        for(var i = 0; i < days.length; i++){
            //first four days
            if(i < 4){
                waffles.push(new this.Waffle(50 + (i * 220), 90, 200, 200, 10, 10,
data, days[i], values, colours));
            }
            //other days
            else{
                waffles.push(new this.Waffle(150 + ((i - 4) * 220), 340, 200, 200,
10, 10, data, days[i], values, colours));
            }
        };
    };

    //call to destroy the chart
    this.destroy = function(){
        waffles.length = 0;
    }

    //draws waffles
    this.draw = function(){
        //draw title
```

```
        fill(0);
        noStroke();
        textAlign('center', 'center');
        textSize(20);
        text(this.title, (50 + (4 * 220) / 2), 40);


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

    //single Waffle
    this.Waffle = function(x, y, width, height, boxes_across, boxes_down, table,
columnHeading, possibleValues, colours){
        //variables
        var x = x;
        var y = y;
        var height = height;
        var width = width;
        var boxes_down = boxes_down;
        var boxes_across = boxes_across;
        var column = table.getColumn(columnHeading);
        var possibleValues = possibleValues;
        var colours = colours;

        //arrays
        var categories = [];
        var boxes =[];
        var label = columnHeading;

        //find the index of a category in the categories array function
        function categoryLocation(categoryName) {
            //check if categoryName matches input categories
            for (var i = 0; i < categories.length; i++){
                if(categoryName == categories[i].name){
                    return i;
                }
            }
            //-1 in case of invalid category names
            return -1;
        }

        //add categories from possibleValues to the categories array
        function addCategories(){
            //loop through all possibe values to add categories
            for(var i = 0; i< possibleValues.length; i++){
                categories.push({
                    "name" : possibleValues[i],
                    "count" : 0,
                    "colour" : colours[i % colours.length]
                })
            }

            //loop through all columns to increment count
            for (var i = 0; i< column.length; i++){
```

```
                var catLocation = categoryLocation(column[i])
                if(catLocation != -1){
                    categories[catLocation].count++
                }
            }

            //iterate over the 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));
            }
        }

        //add boxes function
        function addBoxes(){
            this.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++;
                }
            }
        }

        //add categories and boxes
        addCategories();
        addBoxes();

        //draw chart function
        this.draw = function(){
            stroke(1);
            //chart
            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();
                    }
                }
            }
            //day labels
            push();
            textAlign(CENTER, TOP);
            textSize(16);
            fill(0);
            text(label, x + width/2, y + height + 20);
            pop();
```

```
        }

        //text appear when hovering over chart
        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;
                        }
                    }
                }
            }
        }
    }
}
```

```
function setup() {
    // variables based on waffles
    var days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday'];
    var meals = ['Take-away', 'Cooked from fresh', 'Ready meal', 'Ate out',
'Skipped meal', 'Left overs'];

    // Create a new table
    var table = new p5.Table();

    // Add columns
    for(var i = 0; i < days.length; i++){
        table.addColumn(days[i]);
    }

    // Add 400 rows and populate with random data
    for(var r = 0; r < 400; r++){
        var newRow = table.addRow();
        for(var d = 0; d < days.length; d++){
            var meal;
            // skew the meal selection toward "Take-away" and "Ate out" on
weekdends
            if (days[d] == 'Saturday' || days[d] == 'Sunday') {
                var rand = random();
                if (rand < 0.3) {
                    meal = 'Take-away';
                }
                else if (rand < 0.6) {
                    meal = 'Ate out';
                }
                else {
                    meal = random(meals);
                }
            }
            else {
                meal = random(meals);
            }
            newRow.setString(days[d], meal);
        }
    }

  // Save table as a CSV
  saveTable(table, 'meals.csv');
}
```

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

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

    this.category = category;
    this.mouseOver = function(mouseX, mouseY){
        //is the mouse over this box
        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);
    }
}
```

```
function foodDataUK(){
    //Object Properties
    this.name = 'Food Data in UK';
    this.id = 'food-data-in-uk';
    this.title = 'Food Data in UK';
    this.loaded = false;

    //Variables
    var data;
    var bubbles = [];
    var maxAmt = 0;
    var years = [];
    var yearSlider;
    var minYear;
    var maxYear;

    //preload the csv file
    this.preload = function(){
        var self = this;
        data = loadTable("./data/uk-food/foodData.csv", "csv", "header",
        function(table) {
            self.loaded = true;
        });
    };

    //initialise data and create bubbles and year slider
    this.setup = function(){
        var self = this;
        var rows = data.getRows();
        var numColumns = data.getColumnCount();

        // Assuming the 5th column contains the first year and last column contains
the last year
        minYear = Number(data.columns[5]);
        maxYear = Number(data.columns[numColumns - 1]);

        // Fill the years array with all years in the data
        for(var i = 5; i < numColumns; i++) {
            var y = Number(data.columns[i]);
            years.push(y);
        }

        // Create the year slider
        yearSlider = createSlider(minYear, maxYear, minYear);
        yearSlider.position(400, 10);
        yearSlider.style('width', '800px');
        yearSlider.parent('years'); //html
        yearSlider.input(function() {
            self.changeYear(this.value());
        });

        //to create bubbles from data
        for(var i = 0; i < rows.length; i++){
            if(rows[i].get(0) != ""){
                var units = rows[i].get(4); // Assuming the 5th column contains the
units
                var b = new this.Bubble(rows[i].get(0), units);
                //add data to the bubble for each year
                for(var j = 5; j < numColumns; j++){
```

```
                if(rows[i].get(j) != ""){
                    var n = rows[i].getNum(j);
                    // to find highest value
                    if(n > maxAmt){
                        maxAmt = n;
                    }
                    b.data.push(n);
                }else{
                     b.data.push(0);
                }
            }
            //add bubble to bubbles array
            bubbles.push(b);
        }
    }
    //initialise each bubble data to first year (for comparison)
    for(var i = 0; i < bubbles.length; i++){
        bubbles[i].setData(0);
    }
};

//destroy when changing visualisations
this.destroy = function(){
    bubbles.length = 0;  //empty bubbles array
    yearSlider.remove(); //remove year slider
};

this.draw = function(){
    background(100);
    //centre the coordinates
    translate(width/2, height/2);
    //update and draw each bubble
    textSize(14);
    for(var i = 0; i < bubbles.length; i++)
    {
        bubbles[i].update(bubbles);
        bubbles[i].draw();
    }

    // Reset coordinates
    resetMatrix();

    // Display title and year
    fill(255);
    textSize(18);
    text(this.title,100,40);
    text('Year: ' + yearSlider.value(), 100, 60);
}



this.Bubble = function (_name, _units){
    //bubble properties
    this.size = 20;
    this.target_size = 20;
    this.pos = createVector(0,0);
    this.direction = createVector(0,0);
    this.name = _name;
    this.units = _units;
```

```
        this.color = color(random(40,255), random(40,255), random(40,255));
        this.data = [];
        this.currentData = 0;

        //draw each individual bubble
        this.draw = function()
        {
            push();
            textAlign(CENTER, CENTER);
            noStroke();
            fill(this.color);
            ellipse(this.pos.x, this.pos.y, this.size);
            fill(0);
            // Create a text bounding box based on the bubble size
            var textBoxSize = this.size * 0.8;  // Use 80% of the bubble size for
text
            var lineHeight = 14; // Define line height for text wrap
            var lines = this.calcLines(this.name, textBoxSize);
            var textHeight = lines * lineHeight;
            this.wrapText(this.name, this.pos.x, this.pos.y - textHeight / 2,
textBoxSize, lineHeight);
            if(this.mouseOver()) {
                textSize(16);
                fill(0);
                rect(this.pos.x - 25, this.pos.y + this.size/2 - 20, 50, 20);
                fill(255);
                text(this.data[this.currentData] + ' ' + this.units, this.pos.x,
this.pos.y + this.size/2 - 10);
            }
            pop();
        }

        //update teh bubble position based on other bubbles
        this.update = function(_bubbles){
            this.direction.set(0,0);

            for(var i = 0; i < _bubbles.length; i++){
                if(_bubbles[i].name != this.name){
                    var v = p5.Vector.sub(this.pos,_bubbles[i].pos);
                    var d = v.mag();

                    if(d < this.size/2 + _bubbles[i].size/2){
                        if(d > 0){

                            this.direction.add(v)
                        }else{
                            this.direction.add(p5.Vector.random2D());

                        }
                    }
                }
            }

            this.direction.normalize();
            this.direction.mult(2);
            this.pos.add(this.direction);

            if(this.size < this.target_size){
                this.size += 1;
```

```
            } else if(this.size > this.target_size){
                this.size -= 1;
            }

        }

        //set bubble data and adjust size
        this.setData = function(i){
            this.currentData = i;
            this.target_size = map(this.data[i], 0, maxAmt, 40, 200);
        }

        //check if mouse is over bubble
        this.mouseOver = function(){
            return dist(mouseX - width/2, mouseY - height/2, this.pos.x,
this.pos.y) < this.size/2;
        }

        //calculate hte number of lines need to wrap text
        this.calcLines = function (text, maxWidth){
            let words = text.split(' ');
            let line = '';
            let lines = 1;

            for(let n = 0; n < words.length; n++) {
                let testLine = line + words[n] + ' ';
                let testWidth = textWidth(testLine);

                if(testWidth > maxWidth && n > 0) {
                    line = words[n] + ' ';
                    lines++;
                } else {
                    line = testLine;
                }
            }
            return lines;
        }

        //wrap text within the bubble
        this.wrapText = function(labels, x, y, maxWidth, lineHeight) {
            let words = labels.split(' ');
            let line = '';
            let yoffset = y;

            for(let n = 0; n < words.length; n++) {
                let testLine = line + words[n] + ' ';
                let testWidth = textWidth(testLine);

                if(testWidth > maxWidth && n > 0) {
                    text(line, x, yoffset);
                    line = words[n] + ' ';
                    yoffset += lineHeight;
                } else {
                    line = testLine;
                }
            }
            text(line, x, yoffset);
        }
    }
```

```
    //method to change displayed year (for slider)
    this.changeYear = function(year){
        var y = years.indexOf(year);
        //set to selected year
        for(var i = 0; i < bubbles.length; i++){
            bubbles[i].setData(y);
        }
    }
}
```

```
//the codes below are just an experiment, it does not currently work

//function sgDebt(){
//    //Object Properties
//    this.name = 'SG Debt';
//    this.id = 'sg-debt';
//    this.title = 'SG Debt';
//    this.loaded = false;
//
//    var slider;
//    var data;
//
//    //preload the csv file
//    this.preload = function(){
//        var self = this;
//        data = loadTable("./data/sg-debt/sg-debt.csv", "csv", "header",
//        function(data) {
//            self.loaded = true;
//        });
//    };
//
//    this.setup = function() {
//        // create a slider
//        slider = createSlider(0, data.getColumnCount() - 1, 0);
//        slider.position(10, 10);
//        slider.style('width', '80px');
//    }
//
//    //destroy when changing visualisations
//    this.destroy = function(){
//        yearSlider.remove(); //remove year slider
//    };
//
//    this.draw = function() {
//        background(220);
//
//        let series1 = [];
//        let series2 = [];
//
//        // extract data for each series
//        for (let i = 0; i < data.getRowCount(); i++) {
//            series1.push(data.getNum(i, slider.value()));
//            series2.push(data.getNum(i, slider.value() + 1));
//        }
//
//        // normalise data
//        let maxVal = max(series1.concat(series2));
//        series1 = series1.map(val => map(val, 0, maxVal, 0, height));
//        series2 = series2.map(val => map(val, 0, maxVal, 0, height));
//
//        // draw series 1
//        beginShape();
//        for (let i = 0; i < series1.length; i++) {
//            vertex(i * (width / series1.length), height - series1[i]);
//        }
//        endShape();
//
//        // draw series 2
//        beginShape();
```

```
//         for (let i = 0; i < series2.length; i++) {
//             vertex(i * (width / series2.length), height - series1[i] -
series2[i]);
//         }
//         endShape();
//     }
//}
//
```