data-vis\ai-check.js

```
//------------------------------ START NEW CODE ---------------------------------------
-//
function aiCheck() {
    // Name for the visualisation to appear in the menu bar.
    this.name = 'AI Validation';

    // Each visualisation must have a unique ID with no special characters.
    this.id = 'ai usage';

    // Title to display above the plot.
    this.title = 'how often does people double-check AI generated information!';

    // 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/survey/ai_double_check_survey.csv', 'csv', 'header',
        // Callback function to set the value this.loaded to true.
        function(table) {
            self.loaded = true;
        });
    };

    this.setup = function() {
        // Collecting quantity values from quantity column.
        this.quantity = this.data.getColumn("Quantity").map(Number);
        this.doubleCheck = this.data.getColumn("How often do you double-check AI-generated
information for accuracy?");

        // Create array to store bubbles.
        this.aBubble = [];

        // Loop through each quantity and create bubble object.
        this.createBubbles(9);
    };

    this.destroy = function() {
    };

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


        // Anonymous function to draw text.
        this.drawText();
```

```
50
51          // Loop to create a bubble for every quantity and avoid letting them touch each
    other. NESTED LOOP! O(n²).
52          for (let i = 0; i < this.aBubble.length; i++) {
53              for (let j = 0; j < this.aBubble.length; j++) {
54                  if (i !== j) {
55                      let a = this.aBubble[i];
56                      let b = this.aBubble[j];
57
58                      let dx = a.x - b.x;
59                      let dy = a.y - b.y;
60                      let distBetween = sqrt(dx * dx + dy * dy);
61                      let minDist = (a.size + b.size) / 2;
62
63                      if (distBetween < minDist) {
64                          let angle = atan2(dy, dx);
65                          let overlap = minDist - distBetween;
66
67                          // Pushes bubble i out of bubble j.
68                          a.x += cos(angle) * (overlap / 2) * 20;
69                          a.y += sin(angle) * (overlap / 2) * 20;
70                          b.x -= cos(angle) * (overlap / 2) * 20;
71                          b.y -= sin(angle) * (overlap / 2) * 20;
72                      }
73                  }
74              }
75
76              // Function to create and update each bubble.
77              this.updateAndDrawBubble(i);
78          }
79
80          // Function to draw Legend.
81          this.draw5Legend(
82              450,
83              740,
84              "About half the time",
85              "Always",
86              "Most of the time",
87              "Never",
88              "Sometimes"
89          )
90      }
91
92      // Function to draw the text from the "user", according to the tittle.
93      this.drawText = function() {
94          // Draw user message
95          let message_ai_usage = "Check out "
96
97          push();
98          textSize(30);
99          textAlign(LEFT, TOP);
100         textFont(robotoFont);
101         fill(0);
102         noStroke();
```

```
        text(message_ai_usage, 440, 125.5);
        textFont(robotoFontBold);
        text(this.title, 570, 125.5)
        pop();
    }

    // Function to create the bubbles.
    this.createBubbles = function (sizeValue) {

        // Colors.
        let bubbleColors = ['#FFCD6E', '#688E26', '#1B4D3E', '#C8102E', '#C99A00'];

        // Loop for bubbles.
        for (let i = 0; i < this.data.getRowCount(); i++) {
            let size = this.quantity[i] * sizeValue;

            // Create one bubble object.
            let bubble = {
                x: random(500, 1200),
                y: random(300, 600),
                // x: 400 + i * size * 0.9,
                // y: 400 + i * size * 0.4,
                velocityX: random(-0.2, + 0.2),
                velocityY: random(-0.2, + 0.2),
                size: size,
                color: (bubbleColors[i])
            };

            // Add bubble to array.
            this.aBubble.push(bubble);
        }

    }

    // Function to create and update the bubbles.
    this.updateAndDrawBubble = function (i) {
        // Update position.
        this.aBubble[i].x += this.aBubble[i].velocityX;
        this.aBubble[i].y += this.aBubble[i].velocityY;

        // Interaction with mouse.
        let d = dist(mouseX, mouseY, this.aBubble[i].x, this.aBubble[i].y);
        if (d < this.aBubble[i].size / 2) {
            // Calculate the distance between the mouse and the aBubble (in x and y).
            let dx = this.aBubble[i].x - mouseX;
            let dy = this.aBubble[i].y - mouseY;

            // Calculate the angle of the vector.
            let angle = atan2(dy, dx);

            // Tranform how much to walk in x and y, considering cos to the movement in x and and sin the movement in y.
            this.aBubble[i].x += cos(angle) * 4;
            this.aBubble[i].y += sin(angle) * 4;
```

```
        }

        // Boundary reflection for x on the left.
        if (this.aBubble[i].x < (menuLeft.x + menuLeft.w + 60 + this.aBubble[i].size / 2))
{
            this.aBubble[i].velocityX *= -1;
            this.aBubble[i].x = menuLeft.x + menuLeft.w + 60 + this.aBubble[i].size / 2
        }

        // Boundary reflection for x on the right.
        if (this.aBubble[i].x > (menuLeft.x + menuLeft.w + 60 + width - 420 -
this.aBubble[i].size / 2)) {
            this.aBubble[i].velocityX *= -1;
            this.aBubble[i].x = menuLeft.x + menuLeft.w + 60 + width - 420 -
this.aBubble[i].size / 2;
        }

        // Boundary reflection for y on top.
        if (this.aBubble[i].y < menuTop.h + 80 + this.aBubble[i].size / 2) {
            this.aBubble[i].velocityY *= -1;
            this.aBubble[i].y = menuTop.h + 80 + this.aBubble[i].size / 2;
        }

        // Boundary reflection for y on the bottom.
        if (this.aBubble[i].y > menuTop.w - 805 + menuTop.h + 80 - (this.aBubble[i].size /
2) - 120) {
            this.aBubble[i].velocityY *= -1;
            this.aBubble[i].y = menuTop.w - 805 + menuTop.h + 80 - (this.aBubble[i].size /
2) - 120;
        }

        // Draw.
        fill(this.aBubble[i].color);
        noStroke();
        ellipse(this.aBubble[i].x,
                this.aBubble[i].y,
                this.aBubble[i].size,
                this.aBubble[i].size);

        textAlign(CENTER, CENTER);
        textFont(robotoFont);
        textSize(this.aBubble[i].size * 0.3);
        fill(255);
    text(this.quantity[i] + "%", this.aBubble[i].x, this.aBubble[i].y);
    };

    // Function to draw the legend.
    this.draw5Legend = function (xPos, yPos, labelText1, labelText2, labelText3,
labelText4, labelText5) {
        rightOffset = 300;
        textFont(robotoFont);
        textAlign(LEFT, CENTER);
        textSize(16);
        noFill();
```

```
        noStroke();

        fill('#FFCD6E');
        rect(xPos + rightOffset, yPos, 15, 15, 3);
        fill(0);
        text(labelText1, xPos + 25 + rightOffset, yPos + 6);

        fill('#688E26');
        rect(xPos + 160 + rightOffset, yPos, 15, 15, 3);
        fill(0);
        text(labelText2, xPos + 180 + rightOffset, yPos + 6);

        fill('#1B4D3E');
        rect(xPos + 240 + rightOffset, yPos, 15, 15, 3);
        fill(0);
        text(labelText3, xPos + 260 + rightOffset, yPos + 6);

        fill('#C8102E');
        rect(xPos + 380 + rightOffset, yPos, 15, 15, 3);
        fill(0);
        text(labelText4, xPos + 400 + rightOffset, yPos + 6);

        fill('#C99A00');
        rect(xPos + 460 + rightOffset, yPos, 15, 15, 3);
        fill(0);
        text(labelText5, xPos + 480 + rightOffset, yPos + 6);
    }
}
//------------------------------ END NEW CODE ----------------------------------------//
```

data-vis\canvas_design.js

```
//------------------------------ START NEW CODE ---------------------------------------
-//
  // Class to add all canvas design related functions
  class canvasDesign {
    constructor() {
    }

    // Draw the menu bar (could be on the left, top, right...).
    draw_menu_bar(x, y, w, h, rad, fillColor) {
      push();
      noStroke();
      fill(fillColor);
      rect(x, y, w, h, rad);
      pop();
    }

    // Draw the canvas borders to limit the report.
    draw_canvas_borders(x, y, rectW, rectH, borderWeight, borderColor) {
      push();
      noFill();
      stroke(borderColor);
      strokeWeight(borderWeight);
      rect(x, y, rectW, rectH);
      pop();
    }

    // Draw the background to limit the charts per page.
    draw_canvas_background(x, y, w, h, c, fillColor) {
      push();
      noStroke();
      fill(fillColor);
      rect(x, y, w, h, c);
      pop();
    }

    drawHomeScreen() {
      // Draw user message
      image(user_logo, menuLeft.x + menuLeft.w + 225, 380, 200, 200);
      let welcome_message = "Welcome,"
      let user_message = "user!"
      let user_message2 = "Check the red menu bar on the left"
      let user_message3 = "to start the data exploration!"

      push();
      textSize(60);
      textAlign(LEFT, TOP);
      textFont(robotoFontBold);
      fill(0);
      noStroke();
      text(welcome_message, 750, 400);
      textFont(robotoFont);
      text(user_message, 990, 400);
```

```
      textSize(30);
      text(user_message2, 750, 470);
      text(user_message3, 750, 510);
      pop();
    }
  }
//------------------------------ END NEW CODE ----------------------------------------//
```

data-vis\climate-change.js

```
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.y1AxisLabel = '℃';
  this.y2AxisLabel = "";

  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 * 12,
    rightMargin: width - marginSize - 100,
    topMargin: marginSize * 7,
    bottomMargin: height - marginSize * 4,
    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.
```

```
52  this.preload = function() {
53    var self = this;
54    this.data = loadTable(
55      './data/surface-temperature/surface-temperature.csv', 'csv', 'header',
56      // Callback function to set the value
57      // this.loaded to true.
58      function(table) {
59        self.loaded = true;
60      });
61  };
62
63  this.setup = function() {
64    // Font defaults.
65    textSize(16);
66    textAlign('center', 'center');
67
68    // Set min and max years: assumes data is sorted by year.
69    this.minYear = this.data.getNum(0, 'year');
70    this.maxYear = this.data.getNum(this.data.getRowCount() - 1, 'year');
71
72    // Find min and max temperature for mapping to canvas height.
73    this.minTemperature = min(this.data.getColumn('temperature'));
74    this.maxTemperature = max(this.data.getColumn('temperature'));
75
76    // Find mean temperature to plot average marker.
77    this.meanTemperature = mean(this.data.getColumn('temperature'));
78
79    // Count the number of frames drawn since the visualisation
80    // started so that we can animate the plot.
81    this.frameCount = 0;
82
83    // Create sliders to control start and end years. Default to
84    // visualise full range.
85    this.startSlider = createSlider(this.minYear,
86                                    this.maxYear - 1,
87                                    this.minYear,
88                                    1);
89    this.startSlider.position(400, 10);
90
91    this.endSlider = createSlider(this.minYear + 1,
92                                  this.maxYear,
93                                  this.maxYear,
94                                  1);
95    this.endSlider.position(600, 10);
96  };
97
98  this.destroy = function() {
99    this.startSlider.remove();
100   this.endSlider.remove();
101 };
102
103 this.draw = function() {
104   if (!this.loaded) {
105     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 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.y1AxisLabel,
                   this.y2AxisLabel,
                   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;
    }
```

```
214
215     // Count the number of frames since this visualisation
216     // started. This is used in creating the animation effect and to
217     // stop the main p5 draw loop when all years have been drawn.
218     this.frameCount++;
219
220     // Stop animation when all years have been drawn.
221     if (this.frameCount >= numYears) {
222       //noLoop();
223     }
224   };
225
226   this.mapYearToWidth = function(value) {
227     return map(value,
228                this.startYear,
229                this.endYear,
230                this.layout.leftMargin,   // Draw left-to-right from margin.
231                this.layout.rightMargin);
232   };
233
234   this.mapTemperatureToHeight = function(value) {
235     return map(value,
236                this.minTemperature,
237                this.maxTemperature,
238                this.layout.bottomMargin, // Lower temperature at bottom.
239                this.layout.topMargin);   // Higher temperature at top.
240   };
241
242   this.mapTemperatureToColour = function(value) {
243     var red =  map(value,
244                    this.minTemperature,
245                    this.maxTemperature,
246                    0,
247                    255);
248     var blue = 255 - red;
249     return color(red, 0, blue, 100);
250   };
251 }
252
```

data-vis\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');
```

```
52            }
53
54            var el = select('#' + e.srcElement.id);
55            el.addClass('selected');
56
57            self.selectVisual(e.srcElement.id);
58
59        })
60
61
62        var visMenu = select('.visuals-menu');
63        visMenu.child(menuItem);
64
65        // Preload data if necessary.
66        if (vis.hasOwnProperty('preload')) {
67          vis.preload();
68        }
69      };
70
71      this.findVisIndex = function(visId) {
72        // Search through the visualisations looking for one with the id
73        // matching visId.
74        for (var i = 0; i < this.visuals.length; i++) {
75          if (this.visuals[i].id == visId) {
76            return i;
77          }
78        }
79
80        // Visualisation not found.
81        return null;
82      };
83
84      this.selectVisual = function(visId){
85        var visIndex = this.findVisIndex(visId);
86
87        if (visIndex != null) {
88          // If the current visualisation has a deselect method run it.
89          if (this.selectedVisual != null
90              && this.selectedVisual.hasOwnProperty('destroy')) {
91            this.selectedVisual.destroy();
92          }
93          // Select the visualisation in the gallery.
94          this.selectedVisual = this.visuals[visIndex];
95
96          // Initialise visualisation if necessary.
97          if (this.selectedVisual.hasOwnProperty('setup')) {
98            this.selectedVisual.setup();
99          }
100
101         // Enable animation in case it has been paused by the current
102         // visualisation.
103         loop();
104       }
105     };
```

```
106     }
```

data-vis\helper-functions.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
// ---------------------------------------------------------------------

//------------------------------ START NEW CODE ---------------------------------------
-//
function drawAxis(layout, colour=0, threeAxis=false) {
  stroke(color(colour));
  strokeWeight(1);

  if (threeAxis) {
```

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

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

  // y2-axis
  line(layout.rightMargin,
       layout.topMargin,
       layout.rightMargin,
       layout.bottomMargin);

//------------------------------ END NEW CODE ----------------------------------------//

  }
  else {
   // 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, y1Label, y2Label="", layout) {
  fill(0);
  noStroke();
  textAlign('center', 'center');
  textFont(robotoFont);
  textSize(16);

  // Draw x-axis label.
  text(xLabel,
       (layout.plotWidth() / 2) + layout.leftMargin,
       layout.bottomMargin + (layout.marginSize * 0.25));

  // Draw y-axis label.
  push();
  translate(layout.leftMargin - (layout.marginSize * 0.3),
            layout.bottomMargin / 1.45);
  rotate(- PI / 2);
  text(y1Label, 0, 0);
```

```
106   pop();
107
108   if (y2Label != "") {
109     // Draw y2-axis label.
110     push();
111     translate(layout.rightMargin - (layout.marginSize * 0.3),
112               layout.bottomMargin / 1.45);
113     rotate(- PI / 2);
114     text(y2Label, 0, 80);
115     pop();
116   }
117 }
118
119 function drawYAxisTickLabels(min, max, layout, mapFunction,
120                              decimalPlaces) {
121   // Map function must be passed with .bind(this).
122   var range = max - min;
123   var yTickStep = range / layout.numYTickLabels;
124
125   fill(0);
126   noStroke();
127   textAlign('right', 'center');
128   textFont(robotoFont);
129   textSize(16);
130
131   // Draw all axis tick labels and grid lines.
132   for (i = 0; i <= layout.numYTickLabels; i++) {
133     var value = min + (i * yTickStep);
134     var y = mapFunction(value);
135
136     // Add tick label.
137     text(value.toFixed(decimalPlaces),
138          layout.leftMargin - layout.pad,
139          y);
140
141     if (layout.grid) {
142       // Add grid line.
143       stroke(200);
144       strokeWeight(1);
145       line(layout.leftMargin, y, layout.rightMargin, y);
146     }
147   }
148 }
149
150 //------------------------------ START NEW CODE ---------------------------------------
    -//
151 function drawY2AxisTickLabels(min, max, layout, mapFunction,
152                              decimalPlaces) {
153   // Map function must be passed with .bind(this).
154   var range = max - min;
155   var yTickStep = range / layout.numYTickLabels;
156
157   fill(0);
158   noStroke();
```

```
  textAlign('right', 'center');
  textFont(robotoFont);
  textSize(16);

  // Draw all axis tick labels
  for (i = 0; i <= layout.numYTickLabels; i++) {
    var value = min + (i * yTickStep);
    var y = mapFunction(value);

    // Add tick label.
    text(value.toFixed(decimalPlaces),
         layout.rightMargin + 4 * layout.pad,
         y);
  }
}
//------------------------------ END NEW CODE ----------------------------------------//

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

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

  // Add tick label.
  text(value,
       x,
       layout.bottomMargin + layout.marginSize / 8);

  if (layout.grid) {
    // Add grid line.
    stroke(220);
    strokeWeight(1);
    line(x,
         layout.topMargin,
         x,
         layout.bottomMargin);
  }
}

//------------------------------ START NEW CODE ---------------------------------------
-//
function draw2Legend(xPos, yPos, labelText1, labelText2) {
  textFont(robotoFont);
  textAlign(LEFT, CENTER);
  textSize(16);
  noFill();
  noStroke();

  fill('#C8102E');
  rect(xPos, yPos, 15, 15, 3);
  fill(0);
```

```
  text(labelText1, xPos + 25, yPos + 6);

  fill('#002147');
  rect(xPos + 120, yPos, 15, 15, 3);
  fill(0);
  text(labelText2, xPos + 145, yPos + 6);
}

function drawLegend(xPos, yPos, labels) {
  // Space per block
  let spacing = 145;
  let boxSize = 15;

  // Font
  textFont(robotoFont);
  textAlign(LEFT, CENTER);
  textSize(16);
  noStroke();

  for (let i = 0; i < labels.length; i++) {
    //
    let currentX = xPos + i * spacing;

    //
    fill(labels[i].color);
    rect(currentX, yPos, boxSize, boxSize, 3);

    //
    fill(0);
    text(labels[i].text, currentX + boxSize + 10, yPos + boxSize / 2);
  }
}
//------------------------------ END NEW CODE ----------------------------------------//

```

data-vis\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">

    <!-- ---------------------------- START NEW CODE ---------------------------- -->
    <link href = "https://fonts.googleapis.com/css2?family=Roboto&display=swap"
rel="stylesheet">
    <!-- ---------------------------- END NEW CODE ---------------------------- -->

    <title>Case study 2: Data visualisation</title>

    <!-- Libraries -->
    <script src="lib/p5.min.js"></script>
    <script src="lib/p5.dom.min.js"></script>

    <link rel="stylesheet" href="style.css">

    <!--Canvas-->
    <script src="canvas_design.js"></script>

    <!-- 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="uk-food-attitudes-2018.js"></script>
    <script src="pay-gap-by-job-2017.js"></script>
    <script src="pay-gap-1997-2017.js"></script>
    <script src="climate-change.js"></script>
    <script src="nutrient-intakes.js"></script>
    <script src="ai-check.js"></script>

    <!-- Add extra scripts below -->
    <script src="helper-functions.js"></script>
    <script src="gallery.js"></script>
    <script src="pie-chart.js"></script>
 </head>
  <body>
    <div id="app" class="container">
      <ul class="visuals-menu"></ul>
    </div>
  </body>
</html>
```

data-vis\nutrient-intakes.js

```
//------------------------------ START NEW CODE ---------------------------------------
-//
function NutrientsTimeSeries() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Vitamins Intake TimeSeries';

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

  // Title to display above the plot.
  this.title = 'Vitamins Instake';

    // Names for each axis.
  this.xAxisLabel = 'year';
  this.y1AxisLabel = 'Values';
  this.y2AxisLabel = "";

  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 * 14,
    rightMargin: width - marginSize,
    topMargin: marginSize + 200,
    bottomMargin: height - marginSize * 3,
    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,
  };

```

```
51    // Property to represent whether data has been loaded.
52    this.loaded = false;
53
54    // Preload the data. This function is called automatically by the gallery when a
      visualisation is added.
55    this.preload = function() {
56      var self = this;
57      this.data = loadTable(
58        './data/food/avg-intake-weighted-reference-nutrient-intakes.csv', 'csv', 'header',
59        // Callback function to set the value
60        // this.loaded to true.
61        function(table) {
62          self.loaded = true;
63        });
64
65    };
66
67    this.setup = function() {
68      // Font defaults.
69      textSize(16);
70
71      // Set min and max years: assumes data is sorted by date.
72      this.startYear = Number(this.data.columns[1]);
73      // this.endYear = this.data.getNum(this.data.getRowCount() - 1, 'year');
74      this.endYear = Number(this.data.columns[this.data.columns.length - 1]);
75
76      // Colors
77      for (var i = 0; i < this.data.getRowCount(); i++) {
78        this.colors.push(color(random(0, 255), random(0, 255), random(0, 255)));
79      }
80
81      // Find min and max
82      this.minValue = 0;
83      this.maxValue = 45;
84    };
85
86    this.destroy = function() {
87    };
88
89    this.draw = function() {
90      if (!this.loaded) {
91        console.log('Data not yet loaded');
92        return;
93      }
94
95      // Draw the title above the plot.
96      this.drawTitle();
97
98      // Draw all y-axis labels.
99      drawYAxisTickLabels(this.minValue,
100                          this.maxValue,
101                          this.layout,
102                          this.mapPayGapToHeight.bind(this),
103                          0);
```

```
104
105     // Draw x and y axis.
106     drawAxis(this.layout);
107
108     // Draw x and y axis labels.
109     drawAxisLabels(this.xAxisLabel,
110                    this.y1AxisLabel,
111                    this.y2AxisLabel,
112                    this.layout);
113
114     // Plot all pay gaps between startYear and endYear using the width of the canvas minus
    margins.
115     var previous;
116     var numYears = this.endYear - this.startYear;
117
118     // Loop over all rows and draw a line from the previous value to the current.
119     for (var i = 0; i < this.data.getRowCount(); i++) {
120
121       let row = this.data.getRow(i);
122       let previous = null;
123       let l = row.getString(0);
124
125       for(var j = 1; j < numYears; j++) {
126         var current = {
127           'year': this.startYear + j - 1,
128           'value': row.getNum(j)
129         };
130
131         if (previous != null) {
132           // Draw line segment connecting previous year to current
133           // year pay gap.
134           stroke(this.colors[i]);
135           line(this.mapYearToWidth(previous.year),
136               this.mapPayGapToHeight(previous.value),
137               this.mapYearToWidth(current.year),
138               this.mapPayGapToHeight(current.value));
139
140           // The number of x-axis labels to skip so that only
141           // numXTickLabels are drawn.
142           var xLabelSkip = ceil(numYears / this.layout.numXTickLabels);
143
144           // Draw the tick label marking the start of the previous year.
145           if (i % xLabelSkip == 0) {
146             drawXAxisTickLabel(previous.year, this.layout,
147                                this.mapYearToWidth.bind(this));
148           }
149         }
150         else {
151           noStroke();
152           fill(this.colors[i]);
153           text(l, 630, this.mapPayGapToHeight(current.value))
154         }
155       // 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.minValue,
               this.maxValue,
               this.layout.bottomMargin, // Smaller pay gap at bottom.
               this.layout.topMargin);   // Bigger pay gap at top.
  };
}
//------------------------------ END NEW CODE ----------------------------------------//
```

data-vis\pay-gap-1997-2017.js

```
function PayGapTimeSeries() {

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Gender Pay GAP';

  // 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 chart!';

  // Names for each axis.
  this.xAxisLabel = "Year";
  this.y1AxisLabel = "Percentage of GAP";
  this.y2AxisLabel = "Median per Gender"

  // Define margin
  var marginSize = 140;

  // 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 * 3.2,
    rightMargin: width - marginSize,
    topMargin: marginSize * 2,
    bottomMargin: height - marginSize * 0.8,
    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: 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'));

    // Find min and max average per gender
    this.minMedian = 0;
    this.maxMedianMale = max(this.data.getColumn('median_male'));
    this.maxMedianFemale = max(this.data.getColumn('median_female'));
    this.maxMedian = max(this.maxMedianMale, this.maxMedianFemale);

    // Create variable do draw bars and line smoothly
    this.animationProgress = 0;
  };

  this.destroy = function() {
  };

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

    // Draw text
    this.drawText();

    // Draw all y-axis labels.
    drawYAxisTickLabels(this.minPayGap,
                        this.maxPayGap,
                        this.layout,
                        this.mapPayGapToHeight.bind(this),
                        0);
```

```
103
104     // Draw all y-axis labels.
105     drawY2AxisTickLabels(this.minMedian,
106                          this.maxMedian,
107                          this.layout,
108                          this.mapMedianToHeight.bind(this),
109                          0);
110
111     // Draw x and y axis.
112     drawAxis(this.layout, 0, true);
113
114     // Draw x and y axis labels.
115     drawAxisLabels(this.xAxisLabel,
116                    this.y1AxisLabel,
117                    this.y2AxisLabel,
118                    this.layout);
119
120     // Draw legend
121     this.drawLegend(1100, 740, [
122     {text: "Woman", color: '#C8102E'},
123     {text: "Men", color: '#002147'}
124     ]);
125
126     // Plot all pay gaps between startYear and endYear using the width of the canvas minus
    margins.
127     var previous;
128     var numYears = this.endYear - this.startYear;
129
130     // Loop over all rows and draw a line from the previous value to the current.
131     for (var i = 0; i < this.data.getRowCount(); i++) {
132
133       // Create an object to store data for the current year.
134       var current = {
135         // Convert strings to numbers.
136         'year': this.data.getNum(i, 'year'),
137         'payGap': this.data.getNum(i, 'pay_gap'),
138
139 //------------------------------ START NEW CODE ---------------------------------------
    -//
140         'male': this.data.getNum(i, 'median_male'), // First attribute is row, second is
    column. So: row i, column median_male.
141         'female': this.data.getNum(i, 'median_female')
142 //------------------------------ END NEW CODE ----------------------------------------//
143       };
144
145       if (previous != null) {
146
147 //------------------------------ START NEW CODE ---------------------------------------
    -//
148         if (current.year != 2017) { // NOT DRAWING YEAR 2017 FOR BETTER VISUALIZATION!
149           let barWidth = 25;
150
151           // Draw animation progress for bar
152           this.animationProgress += 0.0010;
```

```
153           this.animationProgress = min(this.animationProgress, 1);
154
155           // Draw bars male
156           fill('#002147');
157           noStroke();
158           rect(this.mapYearToWidth(current.year) - barWidth/2,
159                this.layout.bottomMargin,
160                barWidth,
161                -(this.layout.bottomMargin - this.mapMedianToHeight(current.male)) *
this.animationProgress)
162
163           // Draw bars female
164           fill('#C8102E');
165           noStroke();
166           rect(this.mapYearToWidth(current.year) - barWidth/2,
167               this.layout.bottomMargin,
168               barWidth,
169               -(this.layout.bottomMargin - this.mapMedianToHeight(current.female)) *
this.animationProgress);
170
171           // Draw line segment connecting previous year to current
172           // year pay gap.
173           stroke(0);
174           strokeWeight(5);
175           line(this.mapYearToWidth(previous.year),
176               this.mapPayGapToHeight(previous.payGap),
177               this.mapYearToWidth(current.year),
178               this.mapPayGapToHeight(current.payGap));
179         }
180 //------------------------------ END NEW CODE ---------------------------------------//
181
182         // The number of x-axis labels to skip so that only
183         // numXTickLabels are drawn.
184         var xLabelSkip = ceil(numYears / this.layout.numXTickLabels);
185
186         // Draw the tick label marking the start of the previous year.
187         if (i % xLabelSkip == 0) {
188           drawXAxisTickLabel(previous.year, this.layout,
189           this.mapYearToWidth.bind(this));
190         }
191       }
192
193 //------------------------------ START NEW CODE ---------------------------------------
-//
194       // 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.
195       previous = {
196       year: current.year,
197       payGap: current.payGap,
198       male: current.male,
199       female: current.female
200       };
201 //------------------------------ END NEW CODE ---------------------------------------//
202
```

```
}
  };

  // REMOVED FUNCTION!!!
  this.drawTitle = function() {
    fill(0);
    noStroke();
    textAlign('center', 'center');
    textFont(robotoFont);
    textSize(34);

    text(this.title,
         (this.layout.plotWidth() / 2) + this.layout.leftMargin,
         this.layout.topMargin - (this.layout.marginSize / 2.6));
  };

  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.
  };

//------------------------------ START NEW CODE ---------------------------------------
-//
  this.mapMedianToHeight = function(value) {
    return map(value,
               this.minMedian,
               this.maxMedian,
               this.layout.bottomMargin,
               this.layout.topMargin);
  };

  this.drawText = function() {
    // Draw user message
    let message_pay_gap = "Check out the "

    push();
    textSize(30);
    textAlign(LEFT, TOP);
    textFont(robotoFont);
    fill(0);
    noStroke();
    text(message_pay_gap, 440, 125.5);
    textFont(robotoFontBold);
```

```
    text(this.title, 612, 125.5)
    pop();
    }

  this.drawLegend = function(xPos, yPos, labels) {
  // How to use:
  //   draw5Legend(100, 50, [
  //   { text: "Always", color: '#C8102E' },
  //   { text: "Often", color: '#002147' },
  //   { text: "Sometimes", color: '#FFCD6E' },
  //   { text: "Rarely", color: '#333333' },
  //   { text: "Never", color: '#91A7D2' }
  // ]);


  // Space per block
  let spacing = 145;
  let boxSize = 15;

  // Font
  textFont(robotoFont);
  textAlign(LEFT, CENTER);
  textSize(16);
  noStroke();

  for (let i = 0; i < labels.length; i++) {
    //
    let currentX = xPos + i * spacing;

    //
    fill(labels[i].color);
    rect(currentX, yPos, boxSize, boxSize, 3);

    //
    fill(0);
    text(labels[i].text, currentX + boxSize + 10, yPos + boxSize / 2);
  }
  }
//------------------------------ END NEW CODE ----------------------------------------//
}

```

data-vis\pay-gap-by-job-2017.js

```
function PayGapByJob2017() {

  // Offset for the x/y-axis.
  this.offsetX = 550;
  this.offsetY = 300;
  this.scale =  0.7;

  // Name for the visualisation to appear in the menu bar.
  this.name = 'Pay GAP per Job';

  // 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.padX = this.pad + this.offsetX;

  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.
```

```
52      this.addAxes();
53
54      // Get data from the table object.
55      var jobs = this.data.getColumn('job_subtype');
56      var propFemale = this.data.getColumn('proportion_female');
57      var payGap = this.data.getColumn('pay_gap');
58      var numJobs = this.data.getColumn('num_jobs');
59
60      // Convert numerical data from strings to numbers.
61      propFemale = stringsToNumbers(propFemale);
62      payGap = stringsToNumbers(payGap);
63      numJobs = stringsToNumbers(numJobs);
64
65      // Set ranges for axes.
66      //
67      // Use full 100% for x-axis (proportion of women in roles).
68      var propFemaleMin = 0;
69      var propFemaleMax = 100;
70
71      // For y-axis (pay gap) use a symmetrical axis equal to the
72      // largest gap direction so that equal pay (0% pay gap) is in the
73      // centre of the canvas. Above the line means men are paid
74      // more. Below the line means women are paid more.
75      var payGapMin = -20;
76      var payGapMax = 20;
77
78      // Find smallest and largest numbers of people across all
79      // categories to scale the size of the dots.
80      var numJobsMin = min(numJobs);
81      var numJobsMax = max(numJobs);
82
83      fill(255);
84      stroke(0);
85      strokeWeight(1);
86
87      for (i = 0; i < this.data.getRowCount(); i++) {
88        // Draw an ellipse for each point.
89        // x = propFemale
90        // y = payGap
91        // size = numJobs
92      ellipse(
93        ( map(propFemale[i], propFemaleMin, propFemaleMax,
94              this.pad, width - this.pad)
95          + this.offsetX
96        ) * this.scale,
97
98        ( map(payGap[i], payGapMin, payGapMax,
99              height - this.pad, this.pad)
100         + this.offsetY
101       ) * this.scale,
102
103       map(numJobs[i], numJobsMin, numJobsMax,
104           this.dotSizeMin, this.dotSizeMax)
105       * this.scale
```

```
    );
    }
  };

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

    // Add vertical line.
    line(
      (width/2 + this.offsetX) * this.scale,
      (this.pad           + this.offsetY) * this.scale,
      (width/2 + this.offsetX) * this.scale,
      (height - this.pad  + this.offsetY) * this.scale
    );

    // Add horizontal line.
    line(
      (this.pad + this.offsetX)          * this.scale,
      (height/2 + this.offsetY)          * this.scale,
      (width   - this.pad + this.offsetX) * this.scale,
      (height/2 + this.offsetY)          * this.scale
    );
  };
}

```

data-vis\pie-chart.js

```
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) {

    // 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,
```

```
52          this.diameter, this.diameter,
53          lastAngle, lastAngle + angles[i] + 0.001); // Hack for 0!
54
55      //------------------------------------------------------------------------ START NEW CODE ------------------------------------------------------------------------//
56      // Creating a variable "d" to calculate the distance between two points. Here, it's the distance between the mouse (point 1) and the center of the pie chart (point 2).
57      // Structure: dist(x1, y1, x2, y2), where x1/y1 are coordenates of the first point and x2/y2 coordenats of the second point.
58      let d = dist(mouseX, mouseY, this.x, this.y);
59      let mouse_pie_angle = null;  // ✅ DECLARADA FORA PARA SER USADA DEPOIS
60
61      //
62      if (d < this.diameter / 2) { // Divided by two since radius = diameter/2.
63        // atan2(y, x) is from arctangente, it calculates the angle of a vector between 2 points, where x is the distance in x and y the distance in y. IT'S THE SIZE OF A VECTOR.
64        // It returns an angle in radius from -PI to PI (-180º to +180º)
65        mouse_pie_angle = atan2(mouseY - this.y, mouseX - this.x);
66
67        if (mouse_pie_angle < 0) {
68          mouse_pie_angle += TWO_PI;
69        }
70      }
71
72      if (mouse_pie_angle !== null && mouse_pie_angle >= lastAngle && mouse_pie_angle < lastAngle + angles[i]) {
73        let [r, g, b] = colours[i].levels;
74        let valueText = data[i].toFixed(2) + "%";
75
76        textSize(14);
77        textAlign('left', 'top');
78        textLeading(18);
79
80        let textW = textWidth(valueText) + 10;
81        let textH = 25;
82
83        // Horizontal position
84        let tooltipX = (mouseX + 10 + textW > width)
85                     ? mouseX - textW - 10
86                     : mouseX + 10;
87
88        // Vertical position
89        let tooltipY = mouseY - textH - 10;
90        if (tooltipY < 0) {
91          tooltipY = mouseY + 10;
92        }
93
94        // Background
95        fill(255, 255, 255, 230);
96        stroke(0);
97        strokeWeight(1);
98        rect(tooltipX, tooltipY, textW, textH);
99
100       // Text
```

```
      fill(0);
      noStroke();
      text(valueText, tooltipX + 5, tooltipY + 5);
    }

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

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

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

data-vis\README.md

Case study 3: Data visualisation

Description

This project is an interactive data visualization tool built with **JavaScript**, using the **p5.js** library. It displays multiple datasets through animated and interactive charts focused on themes like:

- Gender Pay Gap
- AI Usage Patterns
- Food Intake and Nutrition
- Climate and Tech Diversity Trends

Each chart is rendered on a custom-designed canvas with accessible color schemes and modular components. The project includes assets, survey data, and UI elements designed for exploratory analysis and educational purposes.

Repository Format

DATA-VISUALIZATION-TOOL-JS/

├── .vscode/

│ └── settings.json

├── assets/

│ ├── roboto_font/

│ ├── ai.png

│ ├── climate.png

│ ├── job.png

│ ├── pay.png

│ ├── tech.png

│ ├── uol_logo.png

│ ├── user_logo.png │ └── vitamin.png

├── data/

│ ├── food/

```
│ │ ├── avg-intake-weighted-reference-nutrient.csv
│ │ ├── avg-intake-weighted-reference-nutrient...
│ │ └── uk-food-attitudes-2018.csv
│ ├── pay-gap/
│ │ ├── sources/
│ │ ├── all-employees-hourly-pay-by-gender.csv
│ │ └── occupation-hourly-pay-by-gender-201...
│ ├── surface-temperature/
│ └── survey/
│ ├── ai_double_check_survey.csv
│ └── ai_usage_survey.csv
├── lib/
│ └── .eslintrc
├── ai-check.js
├── canvas_design.js
├── climate-change.js
├── gallery.js
├── helper-functions.js
├── index.html
├── nutrient-intakes.js
├── pay-gap-1997-2017.js
├── pay-gap-by-job-2017.js
├── pie-chart.js
├── README.md
├── sketch.js
├── style.css
```

├── tech-diversity-gender.js

├── tech-diversity-race.js

├── uk-food-attitudes-2018.js

├── data-vis.zip

├── draft_tasks.xlsx

├── layout-draft.pptx

├── Q2.docx

├── Q2.pdf

├── Q3.docx

├── Q3.pdf

data-vis\sketch.js

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

//------------------------------ START NEW CODE ---------------------------------------//
// Definig global variables for menu left and top
let menuLeft;
let menuTop;

// Acessability
let accessibilityMode = false;

function preload() {
  // Load images
  uol_logo = loadImage('assets/uol_logo.png');
  user_logo = loadImage('assets/user_logo.png');
  ai_icon = loadImage('assets/ai.png');
  climate_icon = loadImage('assets/climate.png');
  job_icon = loadImage('assets/job.png');
  pay_icon = loadImage('assets/pay.png');
  tech_icon = loadImage('assets/tech.png');
  vitamin_icon = loadImage('assets/vitamin.png');

  // Load font
  robotoFont = loadFont('assets/roboto_font/Roboto_Condensed-Regular.ttf');
  robotoFontBold = loadFont('assets/roboto_font/Roboto_Condensed-Bold.ttf');
}
//------------------------------ END NEW CODE ----------------------------------------//

function setup() {
  // Create a canvas to fill the content div from index.html.
  canvasContainer = select('.app');
  var c = createCanvas(1400, 800);
  c.parent('app');

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

//------------------------------ START NEW CODE ---------------------------------------//
  // Margins
  menuLeft = {x: 42, y: 0, w: 260, h: height};
  menuTop  = {x: 42, y: 0, w: width, h: 100};

  // Font
  textFont(robotoFont);

//------------------------------ END NEW CODE ----------------------------------------//

  // Add the visualisation objects here.
  gallery.addVisual(new PayGapTimeSeries());
```

```
  gallery.addVisual(new aiCheck());
  gallery.addVisual(new TechDiversityRace());
  gallery.addVisual(new TechDiversityGender());
  gallery.addVisual(new PayGapByJob2017());
  gallery.addVisual(new ClimateChange());
  gallery.addVisual(new UKFoodAttitudes());
  gallery.addVisual(new NutrientsTimeSeries());
}

function draw() {
//------------------------------ START NEW CODE ---------------------------------------
-//
  // Call constructors.
  canvas = new canvasDesign();

  // Entire canvas background
  background(255);

  // First layer of canvas background
  canvas.draw_canvas_background(menuLeft.x + menuLeft.w, menuTop.h, width, menuTop.w, 0,
"#F2F2F2");

  // Second layer for charts (white)
  canvas.draw_canvas_background(menuLeft.x + menuLeft.w + 60, menuTop.h + 80, width - 420,
menuTop.w - 805, 20, 255);

  // Draw borders into the canvas.
  canvas.draw_canvas_borders(menuLeft.x, 0, width - menuLeft.x, height, 2, 0);

  // Draw menu bar at the left of the canvas.
  canvas.draw_menu_bar(menuLeft.x, menuLeft.y, menuLeft.w, menuLeft.h, 0, "#C8102E");
  // canvas.draw_menu_bar(menuLeft.x, menuLeft.y, menuLeft.w, menuLeft.h, 0, '#E69F00');

  // Draw upper rectangle
  canvas.draw_menu_bar(menuTop.x, menuTop.y, menuTop.w, menuTop.h, 0, "#C7C7C795");

  // Draw UOL logo
  let targetHeight = 70;
  let aspect = uol_logo.width / uol_logo.height;
  let targetWidth = targetHeight * aspect;
  image(uol_logo, 75, 15, targetWidth, targetHeight);

  // Draw tittle
  push();
  textSize(30);
  textAlign(LEFT, TOP);
  textFont(robotoFontBold);
  fill(255);
  noStroke();
  text("MENU BAR", 103, 150);
  pop();

  if (gallery.selectedVisual != null) {
    // Select visuals
```

```
    gallery.selectedVisual.draw();

    // Draw user image
    image(user_logo, menuLeft.x + menuLeft.w + 60, 106, 65, 65);
  }
  else {
    canvas.drawHomeScreen();
  }
}
//------------------------------ END NEW CODE ----------------------------------------//
```

data-vis\style.css

```css
/*----------------------------- START NEW CODE ---------------------------------------*/
body{
    font-family: "Roboto Condensed", sans-serif;
}
/*------------------------------ END NEW CODE ----------------------------------------*/

.menu-item { /*Items from the menu*/
    display: flex;
    align-items: center;
    width: 250px;
    /* width: 200px; */
    height: 50px;
    list-style-type: none;
    padding: 5px;
    color: white
    font-weight: 400;
}

.visuals-menu {
    position: fixed;
    top: 200px;
    left: 10px;
    /* left: 60px; */
}

.visuals {
    margin-left: 200px;
    padding: 20px;
}

.container {
  display: flex;
}

.hover {
    background: #075293;
    color: #ffffff;
}

.selected{
    background: #F2F2F2;
    color: #075293;
    font-weight: 500;
}

```

data-vis\tech-diversity-gender.js

```
function TechDiversityGender() {

  // Offset
  x_offset = 100;
  y_offset = 100;
  // Name for the visualisation to appear in the menu bar.
  this.name = 'Gender Tech Diversity';

  // 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: width - x_offset * 9.5,
    rightMargin: width - x_offset * 1,
    topMargin: 100 + y_offset,
    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('#C8102E');
  this.maleColour = color('#002147');

  // 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(
```

```
52        './data/tech-diversity/gender-2018.csv', 'csv', 'header',
53        // Callback function to set the value
54        // this.loaded to true.
55        function(table) {
56          self.loaded = true;
57        });
58
59    };
60
61    this.setup = function() {
62      // Font defaults.
63      textSize(16);
64    };
65
66    this.destroy = function() {
67    };
68
69    this.draw = function() {
70      if (!this.loaded) {
71        console.log('Data not yet loaded');
72        return;
73      }
74
75      // Draw Female/Male labels at the top of the plot.
76      this.drawCategoryLabels();
77
78      var lineHeight = (height*0.90 - this.layout.topMargin) /
79          this.data.getRowCount();
80
81      for (var i = 0; i < this.data.getRowCount(); i++) {
82
83        // Calculate the y position for each company.
84        var lineY = (lineHeight * i) + this.layout.topMargin;
85
86        // Create an object that stores data from the current row.
87        var company = {
88          // Convert strings to numbers.
89          'name': this.data.getString(i, 'company'),
90          'female': this.data.getNum(i, 'female'),
91          'male': this.data.getNum(i, 'male'),
92        };
93
94        // Draw the company name in the left margin.
95        fill(0);
96        noStroke();
97        textAlign('right', 'top');
98        text(company.name,
99             this.layout.leftMargin - this.layout.pad,
100            lineY);
101
102       // Draw female employees rectangle.
103       fill(this.femaleColour);
104       rect(this.layout.leftMargin,
105            lineY,
```

```
106             this.mapPercentToWidth(company.female),
107             lineHeight - this.layout.pad);
108
109       // Draw male employees rectangle.
110       fill(this.maleColour);
111       rect(this.layout.leftMargin + this.mapPercentToWidth(company.female),
112             lineY,
113             this.mapPercentToWidth(company.male),
114             lineHeight - this.layout.pad);
115     }
116
117     // Draw 50% line
118     stroke(150);
119     strokeWeight(1);
120     line(this.midX,
121          this.layout.topMargin,
122          this.midX,
123          this.layout.bottomMargin);
124
125   };
126
127   this.drawCategoryLabels = function() {
128     fill(0);
129     noStroke();
130     textAlign('left', 'top');
131     text('Female',
132          this.layout.leftMargin,
133          this.layout.pad + 170);
134     textAlign('center', 'top');
135     text('50%',
136          this.midX,
137          this.layout.pad + 170);
138     textAlign('right', 'top');
139     text('Male',
140          this.layout.rightMargin,
141          this.layout.pad + 170);
142   };
143
144   this.mapPercentToWidth = function(percent) {
145     return map(percent,
146                0,
147                100,
148                0,
149                this.layout.plotWidth());
150   };
151 }
152
```

data-vis\tech-diversity-race.js

```
function TechDiversityRace() {

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

  // 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(0.36 * width, 0.18 * height);

    // 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(900, height / 1.7, width * 0.3);

  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);

//------------------------------ START NEW CODE ---------------------------------------
-//
    // Colour to use for each category.
    let colours = [
      color('#002147'), // Royal Blue
      color('#C8102E'), // Scarlet Red
      color('#FFD700'), // Gold
      color('#007A33'), // Emerald Green
      color('#4B0082'), // Deep Purple
      color('#A8A9AD')  // Silver Gray
    ];
//------------------------------ END NEW CODE ----------------------------------------//

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

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

```

data-vis\uk-food-attitudes-2018.js

```
//------------------------------ START NEW CODE ---------------------------------------
-//
function UKFoodAttitudes() {

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

  // Each visualisation must have a unique ID with no special
  // characters.
  this.id = 'uk-food-attitudes';

  // 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/uk-food-attitudes-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(0.36 * width, 0.18 * height);

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

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

  // Create a new pie chart object.
  this.pie = new PieChart(900, height / 1.7, width * 0.3);

```

```
  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 questionType = this.select.value();

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

    // 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.
    let colours = [
      color('#002147'), // Royal Blue
      color('#C8102E'), // Scarlet Red
      color('#FFD700'), // Gold
      color('#007A33'), // Emerald Green
      color('#4B0082') // Deep Purple
    ];

    // Make a title.
    // var title = 'Question: ' + questionType;
    var title = ""

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