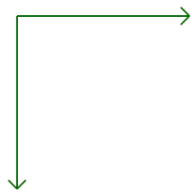
sudo sudo lsof -t -i tcp:8000 | xargs kill -9

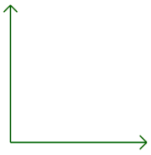
1.1.1 What's the difference between SVG Coordinate Space and Mathematical/ Graph Coordinate Space?

The SVG coordinate system the point x=0, y=0 is the upper left corner. The y-axis is thus reversed compared to a normal graph coordinate system. As y increases in SVG, the points, shapes etc. move down, not up.



In a normal cartesian coordinate system the point x=0, y=0 is at the lower left corner of the graph. As x increases the points move to the right in the coordinate system. As x decreases the points move to the left in the coordinate system. As y increases the points move up in the coordinate system. As y decreases the points move down in the coordinate system.

Here is an illustration of a normal graph coordinate system with 0,0 at the lower left corner:



1.1.2 What is enter() and exit() in d3.js?

The enter function's parameter enter is the enter selection which represents the elements that need to be created. Also, the exit function's parameter exit is the exit selection and contains elements that need to be removed.

1.1.3 What is transform and translate in SVG?
The translate method means it moves the object by x and y. It is assumed to be 0 if y is not provided.
Also, the transform attribute states the list of transform definitions that are applied to an element and its children.

1.1.4 Try to understand the idea of anonymous function and its use in d3.js. If there is a list a = [a,c,b,d,e], what is the return value of this anonymous function: map(function(d,i) {return i+5})

[5,6,7,8,9]

1.1.5 Compare the 2 code snippets below, what will be the output of the 2 code snippets will it be different or same. If it is different, give reasoning on why it is different. (Hint: what does the enter() function does)

Code snippet I:

```
<body>
<p>D3 Tutorials </p>

<script>
    var myData = ['big', 'data', 'assignment', 2, 'submission'];

        var p = d3.select("body")
                  .selectAll("p")
                  .data(myData)
                  .text(function (d, i) {
                      return d;
                  });
</script>
</body>
```

Code snippet II:

```
<body>

<script>
    var data = ['big', 'data', 'assignment', 2, 'submission'];
    var body = d3.select("body")
                 .selectAll("span")
                 .data(data)
                 .enter()
                 .append("span")
                 .text(function(d){return d + " ";});
</script>
</body>
```

The enter function's parameter enter is the enter selection which represents the elements that need to be created.

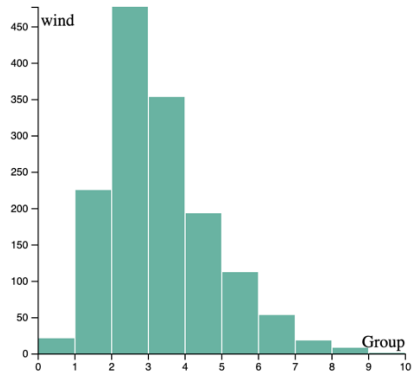
snippet 1 is big

snippet 2 have big data assignment 2 submission in the span element

Yes, the output of the Code snippets is different. When we selected our paragraph elements using .selectAll("p"), it returned only one element in the selection because we had only one paragraph element. Hence, when we bind the data to our paragraph selection, only the first data element is bound to the one available paragraph element. Rest of the data elements from the array were ignored because there were no other <p> elements.

Part II, there are 5 data values in our array. So enter() will create five reference placeholders and append will create five span elements. We provide our data array to the data() function. Since our array has six elements, the code after this will run six times for every element in the array. The enter() function checks for elements corresponding to our five array elements.

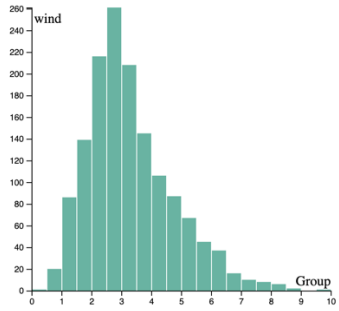
1.2.1



<p> The distribution is mainly around 1 -3, and the highest is around 450.<p>

```
<!-- Create a div where the graph will take place -->
<div id="dataviz"></div>
<p style="background-color:powderblue;"> The distribution is mainly around 1 -3, and the highest is around 450.</p>
<script>

// set the dimensions and margins of the graph
// You can change these values these are just sample values given
var margin = {top: 20, right: 50, bottom: 50, left: 60},
    width = 460 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#dataviz")
  .append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform",
          "translate(" + margin.left + "," + margin.top + ")");

// uncomment the function and complete this function to plot required graphs
// d3.csv("https://github.com/vega/vega/blob/main/docs/data/seattle-weather.csv", function(data) {
d3.csv("seattle-weather.csv", function(data) {
console.log(data[0])
  // X axis: scale and draw:
  var x = d3.scaleLinear()
  // d3.max(data, function(d) { return + d.wind })
      // .domain([0, 1000])     // can use this instead of 1000 to have the max of data: d3.max(data, function(d) { return +
  // var xaxis=   d3.max(data, function(d) { return + d.wind })
      .domain([0,  10])
      .range([0, width]);
  svg.append("g")
      .attr("transform", "translate(0," + height + ")")
      .call(d3.axisBottom(x));

  // set the parameters for the histogram
  var histogram = d3.histogram()
```

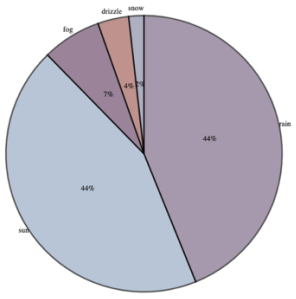
1.2.2



<p> The distribution is mainly around 1 -3, and the highest is around 260.<p>

```
// append the svg object to the body of the page
var svg = d3.select("#dataviz")
  .append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform",
          "translate(" + margin.left + "," + margin.top + ")");

// uncomment the function and complete this function to plot required graphs
// d3.csv("https://github.com/vega/vega/blob/main/docs/data/seattle-weather.csv", function(data) {
d3.csv("seattle-weather.csv", function(data) {
console.log(data[0])
  // X axis: scale and draw:
  var x = d3.scaleLinear()
  // d3.max(data, function(d) { return + d.wind }
      // .domain([0, 1000])      // can use this instead of 1000 to have the max of data: d3.max(data, function(d) { return +d
  // var xaxis=   d3.max(data, function(d) { return + d.wind })
      .domain([0,  10])
      .range([0, width]);
  svg.append("g")
      .attr("transform", "translate(0," + height + ")")
      .call(d3.axisBottom(x));

  // set the parameters for the histogram
  var histogram = d3.histogram()
      .value(function(d) { return d.wind; })   // I need to give the vector of value
      .domain(x.domain())  // then the domain of the graphic
      .thresholds(x.ticks(25)); // then the numbers of bins

  // And apply this function to data to get the bins
  var bins = histogram(data);

  // Y axis: scale and draw:
  var y = d3.scaleLinear()
      .range([height, 0]);
```

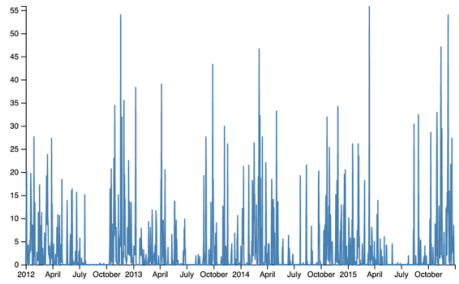
1.2.3



<p> The distribution is mainly occupied by rain and sun. The highest percent is 44%<p>

```
  .domain(data)
  .range(["#98abc5", "#8a89a6", "#7b6888", "#6b486b", "#a05d56"])
  var pie = d3.pie()
    .value(function(d) {return d.value; })
  var data_ready = pie(d3.entries(data))

  // Build the pie chart: Basically, each part of the pie is a path that we build using the arc function.
  svg1
    .selectAll('whatever')
    .data(data_ready)
    .enter()
    .append('path')
    .attr('d', d3.arc()
      .innerRadius(0)
      .outerRadius(radius)
    )
    .attr('fill', function(d){ return(color(d.data.key)) })
    .attr("stroke", "black")
    .style("stroke-width", "2px")
    .style("opacity", 0.7)


  svg1
  .selectAll('mySlices')
  .data(data_ready)
  .enter()
  .append('text')
  .text(function(d){ return d.data.key ; })
  .attr("transform", function(d) { return "translate(" + arcGenerator.centroid(d) + ")";  })
  .style("text-anchor", "middle")
  .style("font-size", 10)

  var arc = d3.arc()
    .innerRadius(0)
    .outerRadius(radius-5)
```

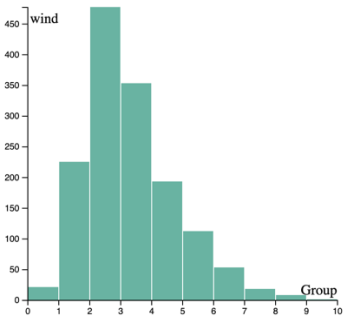
1.2.4



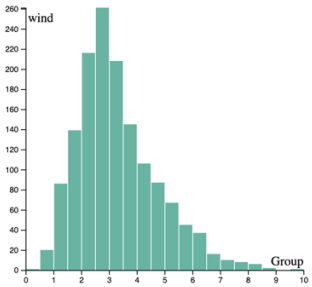
 2015 April and October have the high value around 55.

```
<script>
var margin = {top: 20, right: 50, bottom: 50, left: 60},
    width = 660 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

var svg2 = d3.select("#my_data")
.append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
.append("g")
  .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");
d3.csv("seattle-weather.csv", function(d) {
  return {date: d3.timeParse("%Y-%m-%d")(d.date), value: d.precipitation}
},
function(data){
  var x = d3.scaleTime()
    .domain(d3.extent(data,function(d){ return d.date; }))
    .range([0, width]);
    svg2.append("g")
    .attr("transform", "translate(0,"+ height +")")
    .call(d3.axisBottom(x));


  var y = d3.scaleLinear()
    .domain([0, d3.max(data, function(d){return +d.value; })])
    .range([height, 0]);
    svg2.append("g")
    .call(d3.axisLeft(y));

  svg2.append("path")
    .datum(data)
    .attr("fill","none")
    .attr("stroke","steelblue")
    .attr("stroke-width", 1.5)
    .attr("d", d3.line()
      .x(function(d){return x(d.date)  })
```

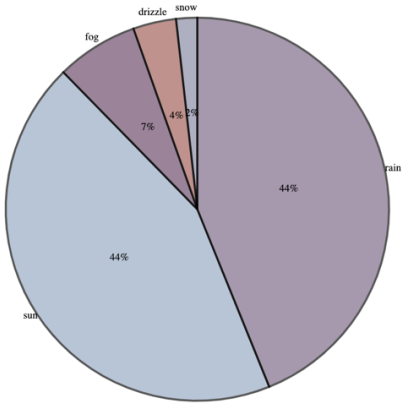
1.2.5 Add a paragraph to the webpage using the tag to note down the main observations from the above obtained plots in 1.2.1, 1.2.2, 1.2.3, 1.2.4 (for eg. distribution, count, important understanding etc).



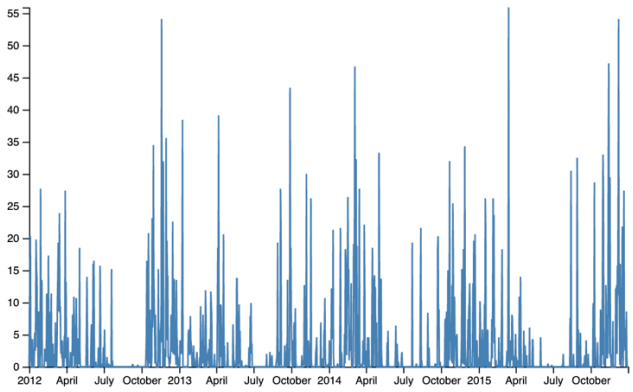
The distribution is mainly around 1 -3, and the highest is around 450.



The distribution is mainly around 1 -3, and the highest is around 260.



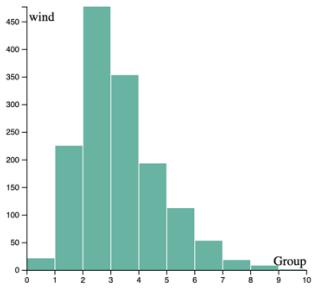
The distribution is mainly occupied by rain and sun. The highest percent is 44%



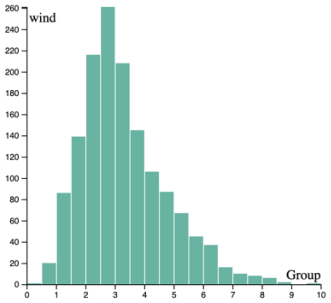
2015 April and October have the high value around 55.

```
<div id="dataviz1"></div>
<p> The distribution is mainly around 1 -3, and the highest is around 450.</p>
<div id="dataviz2"></div>
<p> The distribution is mainly around 1 -3, and the highest is around 260.</p>
<div id="dataviz3"></div>
<p> The distribution is mainly occupied by rain and sun. The highest percent is 44%</p>
<div id="dataviz4"></div>
<p> 2015 April and October have the high value around 55. </p>


<script>

// set the dimensions and margins of the graph
// You can change these values these are just sample values given
var margin = {top: 20, right: 50, bottom: 50, left: 60},
    width = 460 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#dataviz")
  .append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform",
          "translate(" + margin.left + "," + margin.top + ")");

// uncomment the function and complete this function to plot required graphs
// d3.csv("https://github.com/vega/vega/blob/main/docs/data/seattle-weather.csv", function(data) {
d3.csv("seattle-weather.csv", function(data) {
console.log(data[0])
  // X axis: scale and draw:
  var x = d3.scaleLinear()
  // d3.max(data, function(d) { return + d.wind })
      // .domain([0, 1000])    // can use this instead of 1000 to have the max of data: d3.max(data, function(d) { return +d.pr
  // var xaxis=  d3.max(data, function(d) { return + d.wind })
```

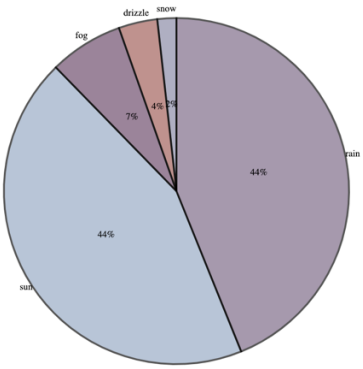
1.2.6 Use the internal style sheet to change the style (font size, font color, indentation) of the paragraph of the web page you created in 1.2.5.

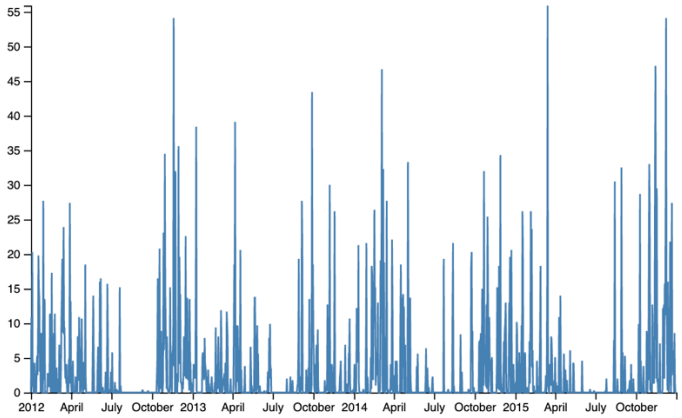


The distribution is mainly around 1 -3, and the highest is around 450.



The distribution is mainly around 1 -3, and the highest is around 260.



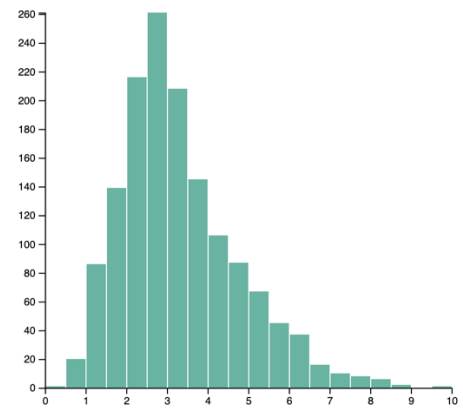


The distribution is mainly occupied by rain and sun. The highest percent is 44%

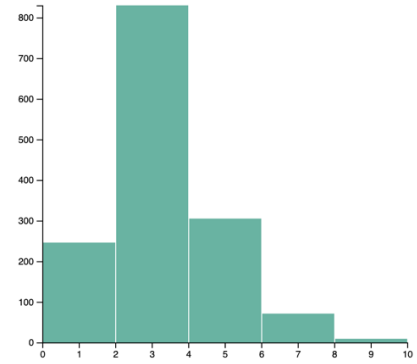
2015 April and October have the high value around 55.

```
<!DOCTYPE html>
<meta charset="utf-8">

<!-- Load d3.js you can use ither version if you want -->
<script src="https://d3js.org/d3.v4.js"></script>
<div id="dataviz1"></div>
<p style="background-color:powderblue;"> The distribution is mainly around 1 -3, and the highest is around 450.</p>
<div id="dataviz2"></div>
<p  style="background-color:powderblue;" > The distribution is mainly around 1 -3, and the highest is around 260.</p>
<div id="dataviz3"></div>
<p  style="color:blue;"> The distribution is mainly occupied by rain and sun. The highest percent is 44%</p>
<div id="dataviz4"></div>
<p  style="color:blue;"> 2015 April and October have the high value around 55. </p>


<script>

// set the dimensions and margins of the graph
// You can change these values these are just sample values given
var margin = {top: 20, right: 50, bottom: 50, left: 60},
    width = 460 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#dataviz")
  .append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform",
          "translate(" + margin.left + "," + margin.top + ")");

// uncomment the function and complete this function to plot required graphs
// d3.csv("https://github.com/vega/vega/blob/main/docs/data/seattle-weather.csv", function(data) {
d3.csv("seattle-weather.csv", function(data) {
console.log(data[0])
  // X axis: scale and draw:
```

1.3.1



bins 25

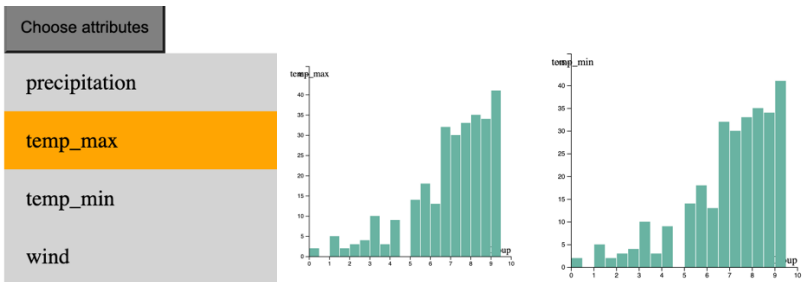


bins 5

```
d3.csv("seattle-weather.csv", function(data) {

  // X axis: scale and draw:
  var x = d3.scaleLinear()
      .domain([0, 10])     // can use this instead of 1000 to have the max of data: d3.max(data, function(d) { return +d.price })
      .range([0, width]);
  svg.append("g")
      .attr("transform", "translate(0," + height + ")")
      .call(d3.axisBottom(x));

  // Y axis: initialization
  var y = d3.scaleLinear()
      .range([height, 0]);
  var yAxis = svg.append("g")

  // A function that builds the graph for a specific value of bin
  function update(nBin) {

    // set the parameters for the histogram
    var histogram = d3.histogram()
        .value(function(d) { return d.wind; })   // I need to give the vector of value
        .domain(x.domain())  // then the domain of the graphic
        .thresholds(x.ticks(nBin)); // then the numbers of bins

    // And apply this function to data to get the bins
    var bins = histogram(data);
    console.log(bins)
    // Y axis: update now that we know the domain
    y.domain([0, d3.max(bins, function(d) { return d.length; })]);   // d3.hist has to be called before the Y axis obviously
    yAxis
        .transition()
        .duration(100)
        .call(d3.axisLeft(y));

    // Join the rect with the bins data
    var u = svg.selectAll("rect")
```

```
    var bins = histogram(data);
    console.log(bins)
    // Y axis: update now that we know the domain
    y.domain([0, d3.max(bins, function(d) { return d.length; })]);   // d3.hist has to be called before the Y axis obviously
    yAxis
        .transition()
        .duration(100)
        .call(d3.axisLeft(y));

    // Join the rect with the bins data
    var u = svg.selectAll("rect")
        .data(bins)

    // Manage the existing bars and eventually the new ones:
    u
        .enter()
        .append("rect") // Add a new rect for each new elements
        .merge(u) // get the already existing elements as well
        .transition() // and apply changes to all of them
        .duration(1000)
          .attr("x", 1)
          .attr("transform", function(d) { return "translate(" + x(d.x0) + "," + y(d.length) + ")"; })
          .attr("width", function(d) { return x(d.x1) - x(d.x0) -1 ; })
          .attr("height", function(d) { return height - y(d.length); })
          .style("fill", "#69b3a2")


    // If less bar in the new histogram, I delete the ones not in use anymore
    u
        .exit()
        .remove()

    }
```

1.3.2




```
      .append("path")
        .datum(data)
        .attr("d", d3.line()
          .x(function(d) { return x(+d.time) })
          .y(function(d) { return y(+d.valueA) })
        )
        .attr("stroke", function(d){ return myColor("valueA") })
        .style("stroke-width", 4)
        .style("fill", "none")

    // A function that update the chart
    function update(selectedGroup) {

      // Create new data with the selection?
      var dataFilter = data.map(function(d){return {time: d.time, value:d[selectedGroup]} })

      // Give these new data to update line
      line
          .datum(dataFilter)
          .transition()
          .duration(1000)
          .attr("d", d3.line()
            .x(function(d) { return x(+d.time) })
            .y(function(d) { return y(+d.value) })
          )
          .attr("stroke", function(d){ return myColor(selectedGroup) })
    }

    // When the button is changed, run the updateChart function
    d3.select("#selectButton").on("change", function(d) {
        // recover the option that has been chosen
        var selectedOption = d3.select(this).property("value")
        // run the updateChart function with this selected option
        update(selectedOption)
    })
```

Part II Tasks: (60 marks)
2.1

mpg	cyl	displ	hp	weight	accel	yr	origin	name
18	8	307	130	3504	12	70	1	chevrolet chevelle malibu
15	8	350	165	3693	11.5	70	1	buick skylark 320
18	8	318	150	3436	11	70	1	plymouth satellite
16	8	304	150	3433	12	70	1	amc rebel sst

```
$(document).ready(function(){
 $('#load_data').click(function(){
  $.ajax({
   url:"auto-mpg.csv",
   dataType:"text",
   success:function(data)
   {
    var data = data.split(/\r?\n|\r/);
    var table_data = '<table class="table table-bordered table-striped">';
    for(var count = 0; count< 6; count++)
    {
     var cell_data = data[count].split(",");
     table_data += '<tr>';
     for(var cell_count=0; cell_count<cell_data.length; cell_count++)
     {
      if(count === 0)
      {
       table_data += '<th>'+cell_data[cell_count]+'</th>';
      }
      else
      {
       table_data += '<td>'+cell_data[cell_count]+'</td>';
      }
     }
     table_data += '</tr>';
    }
    table_data += '</table>';
    // console.log(table_data)
    $('#table').html(table_data);
   }
  });
```

2.1.2

each model year number of cars	number of cars
70	29
71	28
72	28
73	40
74	27
75	30
76	34
77	28
78	29
79	29
80	29
81	31

```
<script>
   data = {70:29, 71:28, 72:28,73:40, 74:27, 75:30, 76:34  , 77:28 , 78:29 , 79:29, 80: 29, 81 : 31 }
    function showTableData() {
        document.getElementById('info').innerHTML = "";
        var myTab = document.getElementById('empTable');

        // LOOP THROUGH EACH ROW OF THE TABLE AFTER HEADER.
        for (i = 1; i < myTab.rows.length; i++) {

            // GET THE CELLS COLLECTION OF THE CURRENT ROW.
            var objCells = myTab.rows.item(i).cells;

            // LOOP THROUGH EACH CELL OF THE CURENT ROW TO READ CELL VALUES.
            for (var j = 0; j < objCells.length; j++) {
                info.innerHTML = info.innerHTML + ' ' + objCells.item(j).innerHTML;
            }
            info.innerHTML = info.innerHTML + '<br />';      // ADD A BREAK (TAG).
        }
    }
</script>
</html>
```

2.1.3

modelyear	cylinder
70	196
71	156
72	163
73	255
74	142
75	168
76	153
77	153
78	193
79	169
80	120
81	134
82	130

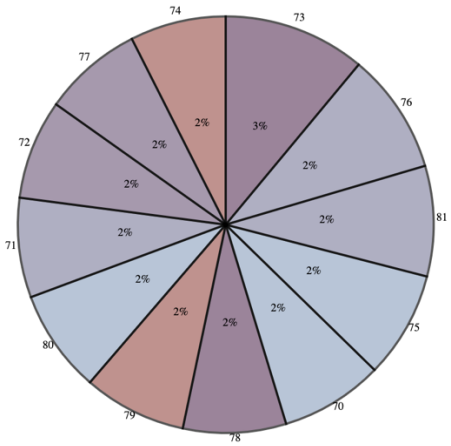
```
<script>
$(document).ready(function(){
 $('#load_data').click(function(){
  $.ajax({
   url:"2.1.4.csv",
   dataType:"text",
   success:function(data)
   {
    var data = data.split(/\r?\n|\r/);
    var table_data = '<table class="table table-bordered table-striped">';
    for(var count = 0; count<data.length; count++)
    {
     var cell_data = data[count].split(" ");
     table_data += '<tr>';
     for(var cell_count=0; cell_count<cell_data.length; cell_count++)
     {
      if(count === 0)
      {
       table_data += '<th>'+cell_data[cell_count]+'</th>';
      }
      else
      {
       table_data += '<td>'+cell_data[cell_count]+'</td>';
      }
     }
     table_data += '</tr>';
    }
    table_data += '</table>';
    // console.log(table_data)
    $('#employee_table').html(table_data);
   }
  });
 });

});
```

2.1.4

acceleration	numberofcar
8	1
8.5	2
9	1
9.5	2
10	4
10.5	1
11	7
11.1	1
11.2	1
11.3	1
11.4	2
11.5	7
11.6	1
12	10
12.1	1
12.2	2

```
<script>
$(document).ready(function(){
 $('#load_data').click(function(){
  $.ajax({
   url:"2.1.3_cur.csv",
   dataType:"text",
   success:function(data)
   {
    var data = data.split(/\r?\n|\r/);
    var table_data = '<table class="table table-bordered table-striped">';
    for(var count = 0; count<data.length; count++)
    {
     var cell_data = data[count].split(",");
     table_data += '<tr>';
     for(var cell_count=0; cell_count<cell_data.length; cell_count++)
     {
      if(count === 0)
      {
       table_data += '<th>'+cell_data[cell_count]+'</th>';
      }
      else
      {
       table_data += '<td>'+cell_data[cell_count]+'</td>';
      }
     }
     table_data += '</tr>';
    }
    table_data += '</table>';
    // console.log(table_data)
    $('#employee_table').html(table_data);
   }
  });
 });

});
</script>
```

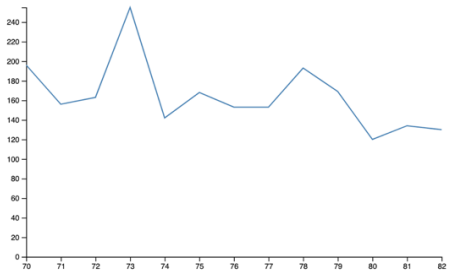
2.2.1




```
var color = d3.scaleOrdinal()
  .domain(data)
  .range(["#98abc5", "#8a89a6", "#7b6888", "#6b486b", "#a05d56"])
var pie = d3.pie()
  .value(function(d) {return d.value; })
var data_ready = pie(d3.entries(data))

// Build the pie chart: Basically, each part of the pie is a path that we build using the arc function.
svg1
  .selectAll('whatever')
  .data(data_ready)
  .enter()
  .append('path')
  .attr('d', d3.arc()
    .innerRadius(0)
    .outerRadius(radius)
  )
  .attr('fill', function(d){ return(color(d.data.key)) })
  .attr("stroke", "black")
  .style("stroke-width", "2px")
  .style("opacity", 0.7)


svg1
.selectAll('mySlices')
.data(data_ready)
.enter()
.append('text')
.text(function(d){ return d.data.key ; })
.attr("transform", function(d) { return "translate(" + arcGenerator.centroid(d) + ")";  })
.style("text-anchor", "middle")
.style("font-size", 10)

var arc = d3.arc()
  .innerRadius(0)
  .outerRadius(radius-5)
```

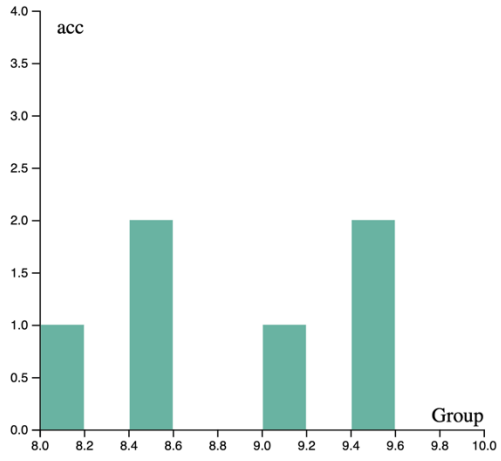
2.2.2

Using the array created in 2.1.3, plot a line graph using D3 along with the table from 2.1.3 to see the total number of cylinders for each model year, label the x and y axis and sort the x axis in decreasing order




```
var svg2 = d3.select("#my_data")
.append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
.append("g")
  .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");
d3.csv("2.1.3_car.csv", function(d) {
  console.log(d)
  return {date: d.modelyear, value: d.cylinder}
},
function(data){
    var x = d3.scaleLinear()
    .domain(d3.extent(data,function(d){ return d.date; }))
    .range([0, width]);
    svg2.append("g")
    .attr("transform", "translate(0,"+ height +")")
    .call(d3.axisBottom(x));


  var y = d3.scaleLinear()
    .domain([0, d3.max(data, function(d){return +d.value; })])
    .range([height, 0]);
    svg2.append("g")
    .call(d3.axisLeft(y));

  svg2.append("path")
    .datum(data)
    .attr("fill","none")
    .attr("stroke","steelblue")
    .attr("stroke-width", 1.5)
    .attr("d", d3.line()
      .x(function(d){return x(d.date)  })
      .y(function(d){return y(d.value) })
      )
})
```

2.2.3

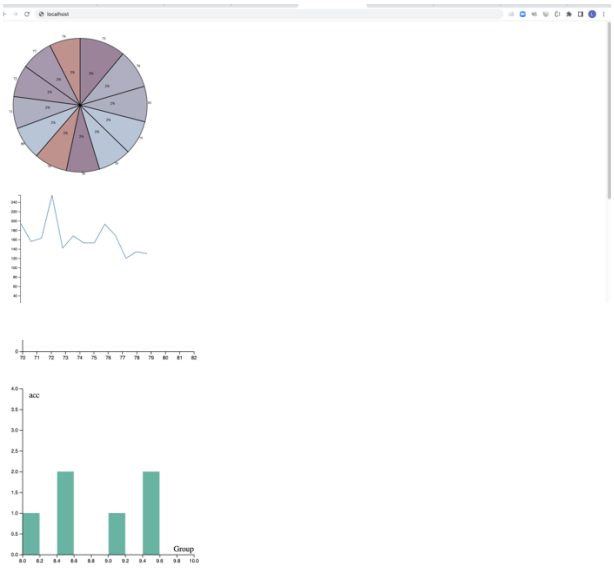


```
var histogram = d3.histogram()
    .value(function(d) { return d.accel; })   // I need to give the vector of value
    .domain(x.domain())  // then the domain of the graphic
    .thresholds(x.ticks(10)); // then the numbers of bins

// And apply this function to data to get the bins
var bins = histogram(data);

// Y axis: scale and draw:
var y = d3.scaleLinear()
    .range([height, 0]);
    y.domain([0, d3.max(bins, function(d) { return d.length; })]);   // d3.hist has to be called befo

svg.append("g")
    .call(d3.axisLeft(y));
    // append the bar rectangles to the svg element
svg.append("text")
    .attr("class", "x label")
    .attr("text-anchor", "end")
    .attr("x", width)
    .attr("y", height - 6)
    .text("Group");
svg.append("text")
  .attr("class", "y label")
  .attr("text-anchor", "end")
  .attr("x", 35)
  .attr("y", 6)
  .attr("dy", ".75em")
  // .attr("transform", "rotate(-90)")
  .text("acc");
svg.selectAll("rect")
    .data(bins)
    .enter()
    .append("rect")
      .attr("x", 1)
      .attr("transform", function(d) { return "translate(" + x(d.x0) + "," + y(d.length) + ")"; })
```

2.3.1 Follow the tutorial for installing Apache HTTP server
&2.3.2 Create a Virtual Host

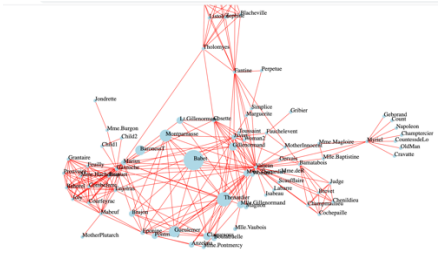
```
larry_1@liangdeMBP html % sudo apachectl start
Password:
larry_1@liangdeMBP html %
```

2.3.3 Host your webpage (webpage from Part II problem 2) on the virtual host.

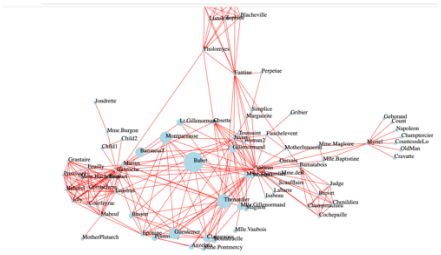


Part III Tasks: (30 marks)

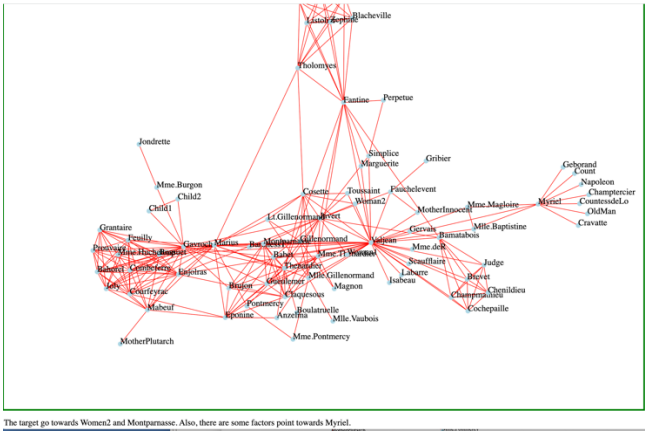
3.1.1 Plot a Network graph that shows character co-occurrence in Les &3.1.2 Add labels to each node. (i.e. each node should show the name of the character)



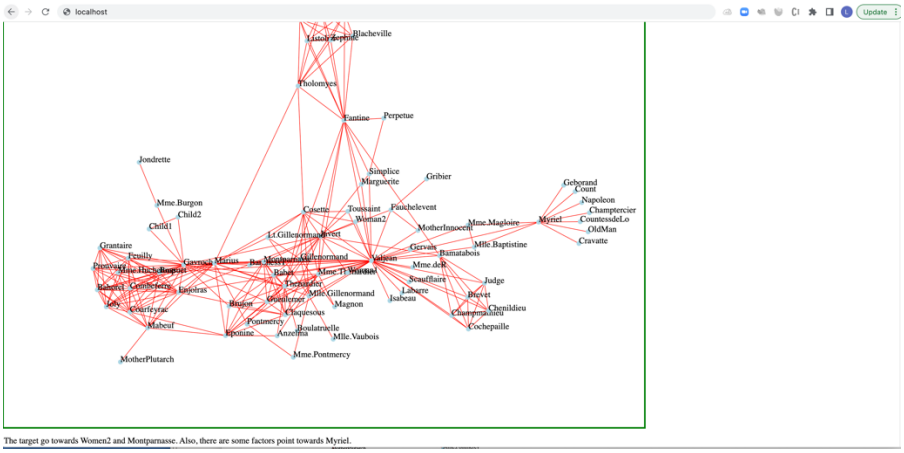
3.1.3 It can be difficult to observe micro and macro features simultaneously with complex graphs. If you zoom in for detail, the graph is too big to view in its entirety. Use fisheye distortion (magnifies the local region around the mouse, while leaving the larger graph unaffected for context) for the above network graph. (you can use D3's fisheye plugin).

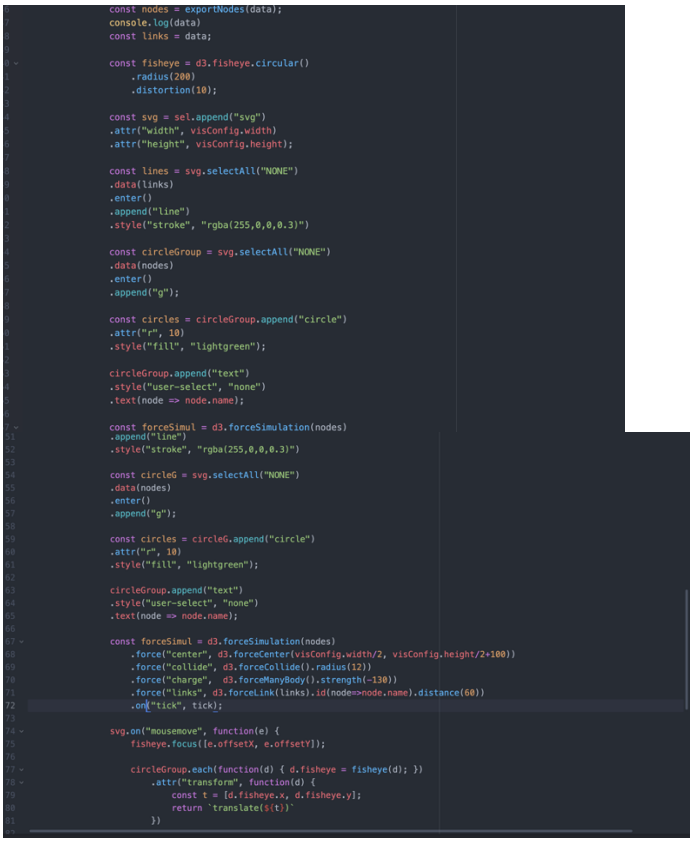


3.2.1 Add a <p> tag to your HTMl file and infer what you see from the graph.



3.2.2 Host your webpage (from 3.1.3) on the virtual host.





```
const nodes = exportNodes(data);
console.log(data)
const links = data;

const fisheye = d3.fisheye.circular()
    .radius(200)
    .distortion(10);

const svg = sel.append("svg")
.attr("width", visConfig.width)
.attr("height", visConfig.height);

const lines = svg.selectAll("NONE")
.data(links)
.enter()
.append("line")
.style("stroke", "rgba(255,0,0,0.3)")

const circleGroup = svg.selectAll("NONE")
.data(nodes)
.enter()
.append("g");

const circles = circleGroup.append("circle")
.attr("r", 10)
.style("fill", "lightgreen");

circleGroup.append("text")
.style("user-select", "none")
.text(node => node.name);

const forceSimul = d3.forceSimulation(nodes)
.append("line")
.style("stroke", "rgba(255,0,0,0.3)")

const circleG = svg.selectAll("NONE")
.data(nodes)
.enter()
.append("g");

const circles = circleG.append("circle")
.attr("r", 10)
.style("fill", "lightgreen");

circleGroup.append("text")
.style("user-select", "none")
.text(node => node.name);

const forceSimul = d3.forceSimulation(nodes)
    .force("center", d3.forceCenter(visConfig.width/2, visConfig.height/2+100))
    .force("collide", d3.forceCollide().radius(12))
    .force("charge", d3.forceManyBody().strength(-130))
    .force("links", d3.forceLink(links).id(node=>node.name).distance(60))
    .on("tick", tick);

svg.on("mousemove", function(e) {
    fisheye.focus([e.offsetX, e.offsetY]);

    circleGroup.each(function(d) { d.fisheye = fisheye(d); })
        .attr("transform", function(d) {
            const t = [d.fisheye.x, d.fisheye.y];
            return `translate(${t})`
        })
```