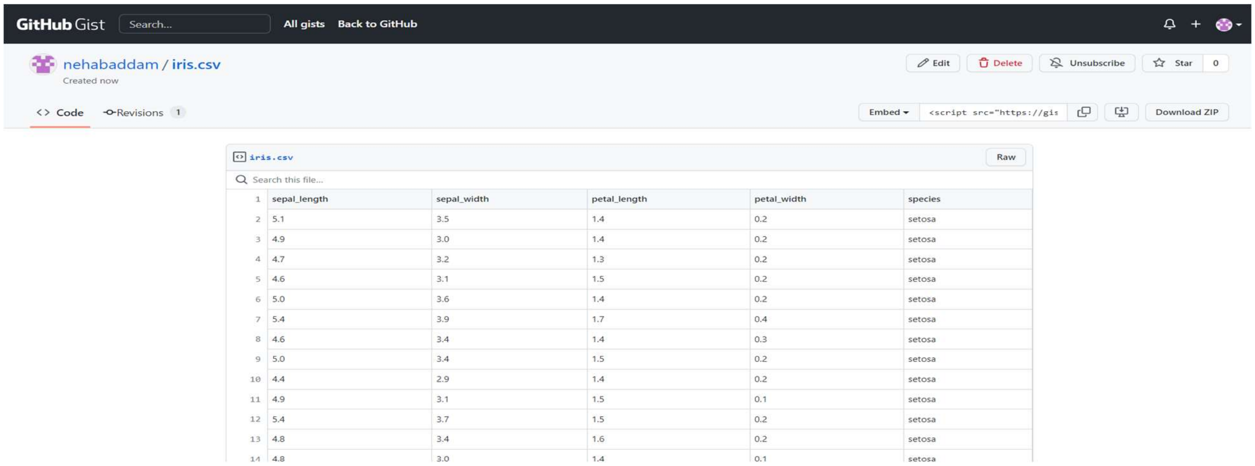
CSCE 5320 Scientific Data Visualization
ICE-7
Common Visualization Idioms

1. I have downloaded the Iris data and created a reusable scatter plot.

Firstly, I downloaded the Iris data file and created a public gist, and added the data to the gist. I shall use this data to create a scatter plot.

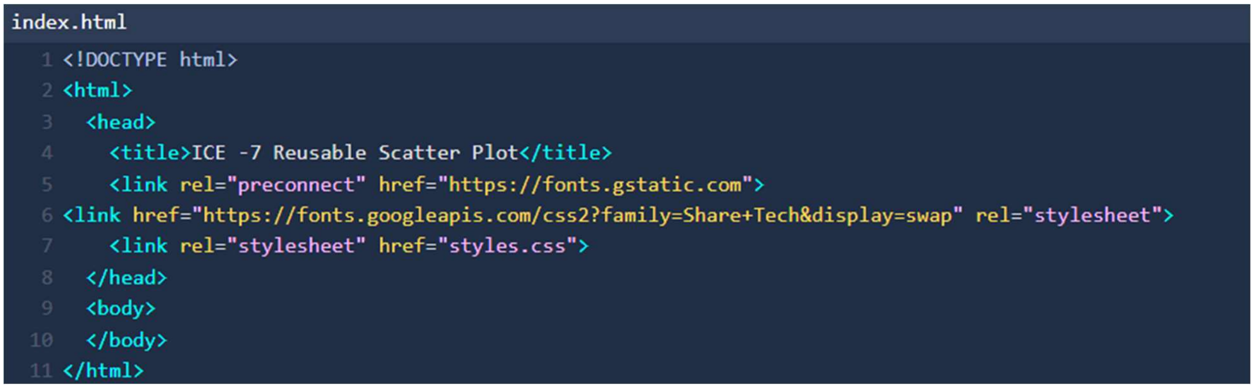


sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa

Creating a reusable scatter plot with each iris species indicated using a symbol.

Index.html:

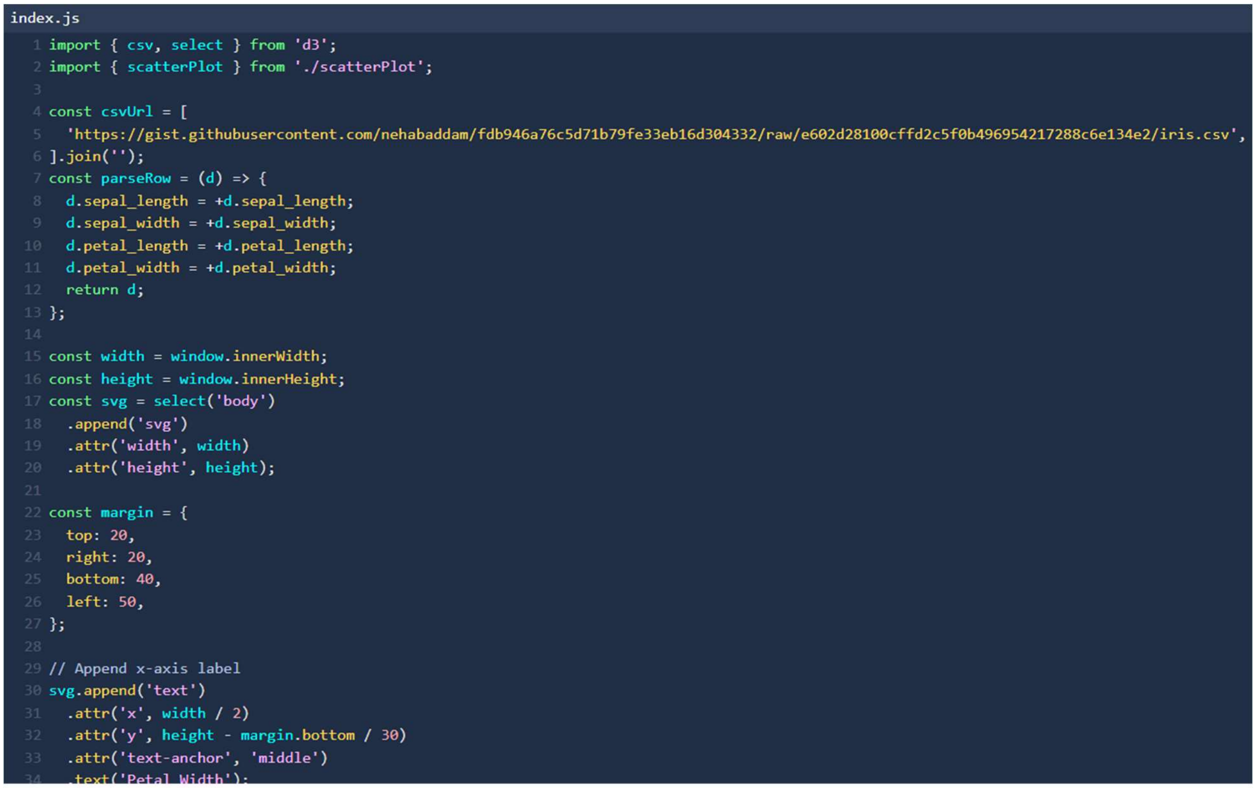
Firstly, I have created an HTML page. The <head> section includes a <title> tag which sets the page title to "ICE -7 Reusable Scatter Plot." The <link> tag within the <head> element reconnects to the Google Fonts API and then connects to a font stylesheet that contains the "Share Tech" font family. The <link> tag for the CSS file named "styles.css" is also included in the <head> section. The <body> section is currently empty, indicating that no content is displayed on the page. The goal of this HTML document is to establish a structure for a webpage that can display the reusable scatter plot chart that we are going to make using D3.js with the iris dataset.


index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>ICE -7 Reusable Scatter Plot</title>
    <link rel="preconnect" href="https://fonts.gstatic.com">
<link href="https://fonts.googleapis.com/css2?family=Share+Tech&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
  </body>
</html>
```

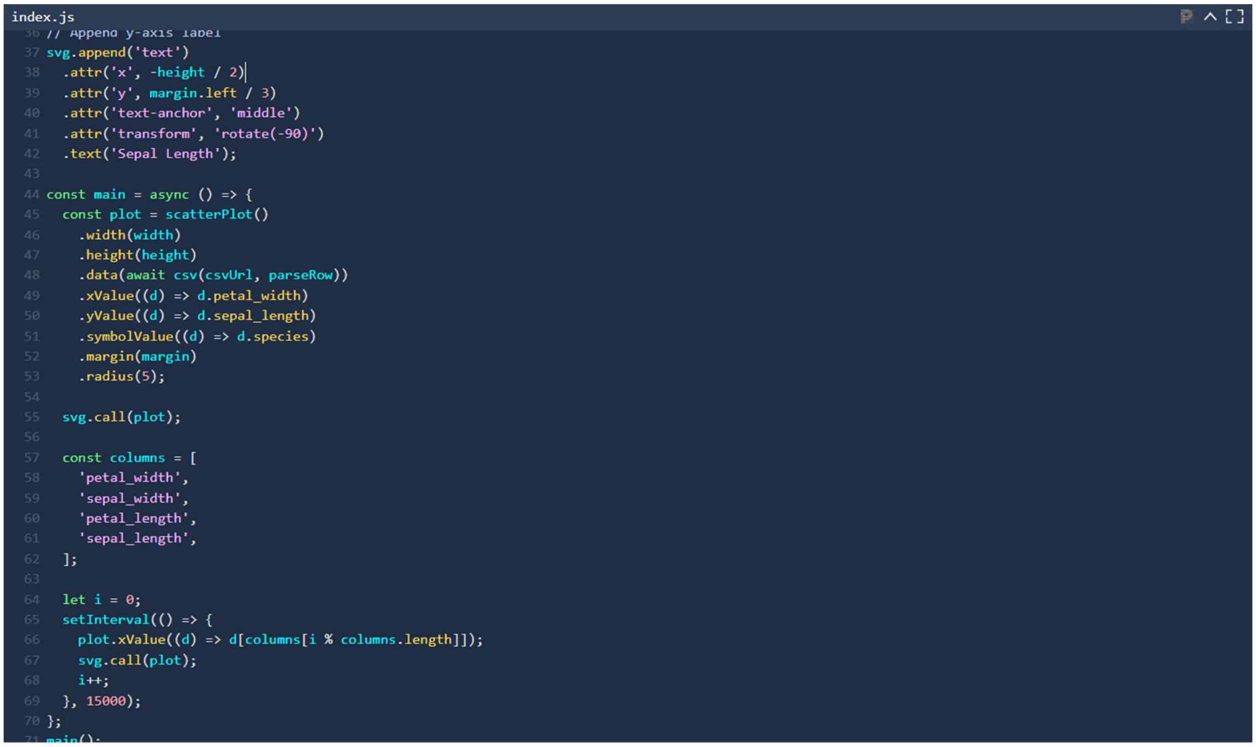
Index.js:

Secondly, I have created this JavaScript program that utilizes the D3.js library to create a scatter plot visualization of the relationship between the sepal length and petal width of iris flowers. The modules such as CSV and select from D3.js library are imported. A custom module scatterPlot is used to create the scatter plot. The iris dataset is converted to the desired format using the parseRow function, which changes the numerical columns into numeric data types. The chart's width and height are set to the browser window dimensions, and an SVG element is appended to the HTML page's body. The scatterPlot function is called with its properties set. The data is retrieved from the CSV file using the CSV function. The x-axis and y-axis values are set to petal width and sepal length, respectively, while the symbol value function is used to set the symbol type of each point to its species. The svg.call(plot) function is called to display the scatter plot on the SVG element. The x-axis values are then updated dynamically every 15 seconds to show the relationship between sepal width and petal length. An array of column names and a variable i that increments every 15 seconds are defined. The x-value function of the scatterPlot function is updated with a new column name from the array, which is selected using the modulo operator. Finally, the svg.call(plot) function is used to update the chart with the new x-axis values. This code is used to create interactive scatter plots using actual data. It is adaptable to visualize different datasets and types of charts.



index.js

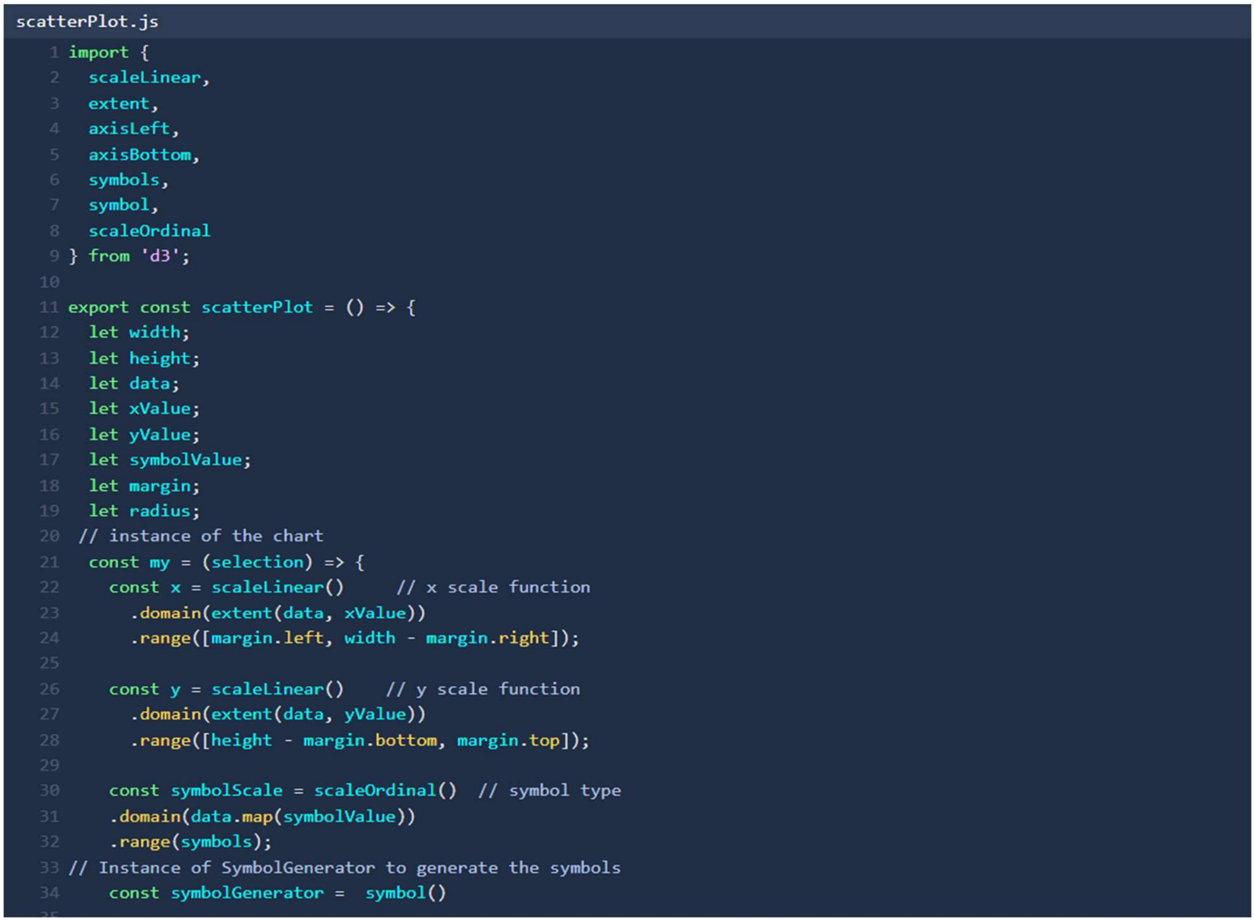
```javascript
import { csv, select } from 'd3';
import { scatterPlot } from './scatterPlot';

const csvUrl = [
  'https://gist.githubusercontent.com/nehabaddam/fdb946a76c5d71b79fe33eb16d304332/raw/e602d28100cffd2c5f0b496954217288c6e134e2/iris.csv',
].join('');
const parseRow = (d) => {
  d.sepal_length = +d.sepal_length;
  d.sepal_width = +d.sepal_width;
  d.petal_length = +d.petal_length;
  d.petal_width = +d.petal_width;
  return d;
};

const width = window.innerWidth;
const height = window.innerHeight;
const svg = select('body')
  .append('svg')
  .attr('width', width)
  .attr('height', height);

const margin = {
  top: 20,
  right: 20,
  bottom: 40,
  left: 50,
};

// Append x-axis label
svg.append('text')
  .attr('x', width / 2)
  .attr('y', height - margin.bottom / 30)
  .attr('text-anchor', 'middle')
  .text('Petal Width');
```



```
// Append y-axis label
svg.append('text')
  .attr('x', -height / 2)
  .attr('y', margin.left / 3)
  .attr('text-anchor', 'middle')
  .attr('transform', 'rotate(-90)')
  .text('Sepal Length');

const main = async () => {
  const plot = scatterPlot()
    .width(width)
    .height(height)
    .data(await csv(csvUrl, parseRow))
    .xValue((d) => d.petal_width)
    .yValue((d) => d.sepal_length)
    .symbolValue((d) => d.species)
    .margin(margin)
    .radius(5);

  svg.call(plot);

  const columns = [
    'petal_width',
    'sepal_width',
    'petal_length',
    'sepal_length',
  ];

  let i = 0;
  setInterval(() => {
    plot.xValue((d) => d[columns[i % columns.length]]);
    svg.call(plot);
    i++;
  }, 15000);
};
main();
```

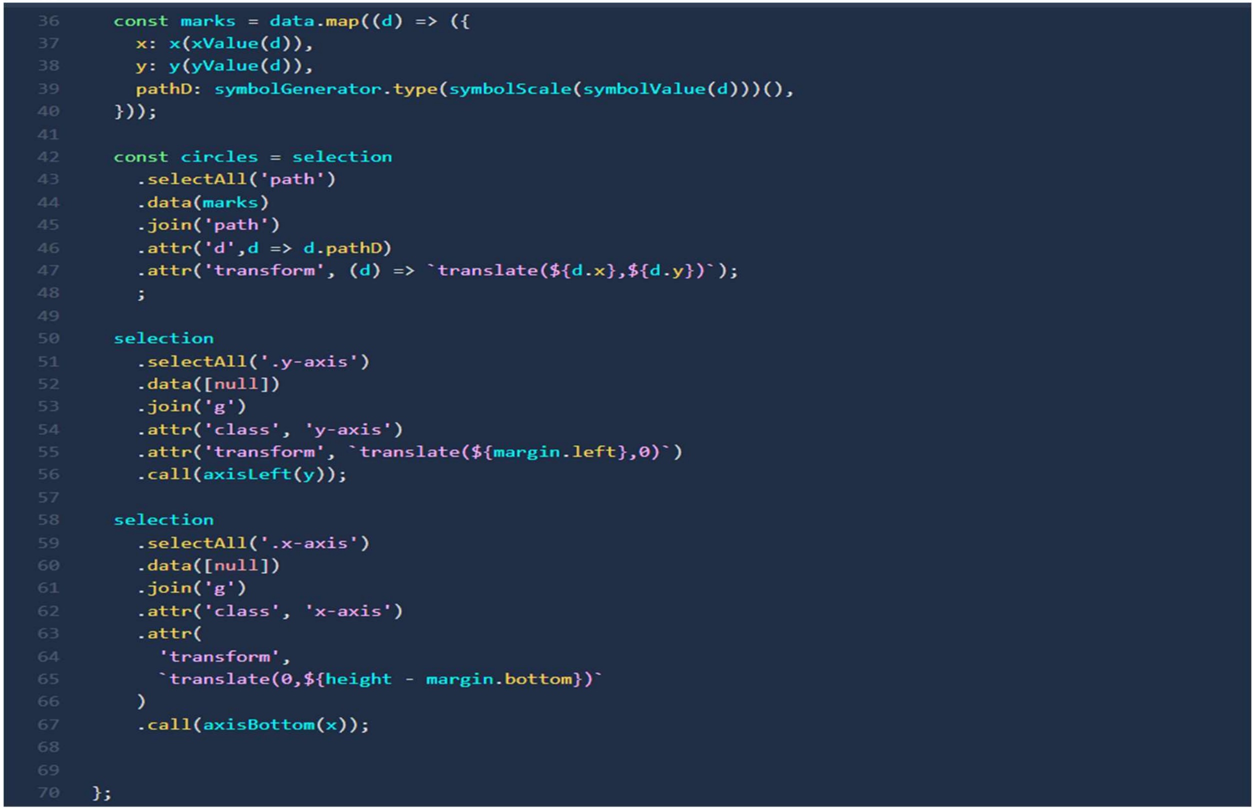
<u>Scatterplot.js:</u>

Finally, we create this module that was imported and used in the index.js file. This module is used to define a function called scatterPlot, which utilizes the D3.js library to create a scatter plot. This function can be used repeatedly to generate scatter plots for various data sets. It has several setter functions, including width, height, data, xValue, yValue, symbolValue, margin, and radius, which can be used to customize the chart generated by the function. The chart is generated by creating scales for the x and y axes, symbol scales for different symbols, and generating SVG paths for each data point to render them. The chart also includes x and y-axis labels generated using axisBottom and axisLeft functions.

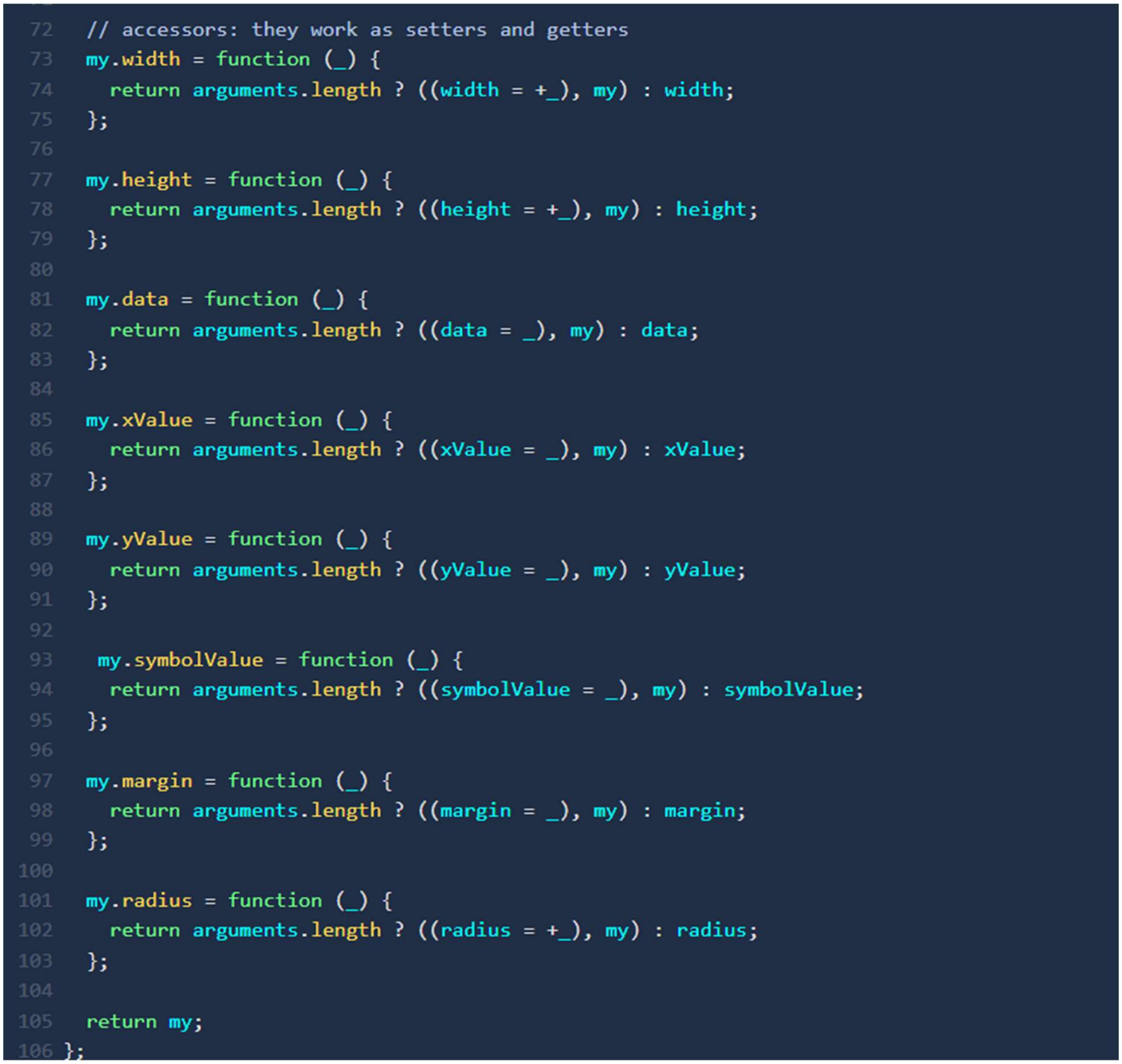


scatterPlot.js

```
import {
  scaleLinear,
  extent,
  axisLeft,
  axisBottom,
  symbols,
  symbol,
  scaleOrdinal
} from 'd3';

export const scatterPlot = () => {
  let width;
  let height;
  let data;
  let xValue;
  let yValue;
  let symbolValue;
  let margin;
  let radius;
 // instance of the chart
  const my = (selection) => {
    const x = scaleLinear()     // x scale function
      .domain(extent(data, xValue))
      .range([margin.left, width - margin.right]);

    const y = scaleLinear()    // y scale function
      .domain(extent(data, yValue))
      .range([height - margin.bottom, margin.top]);

    const symbolScale = scaleOrdinal()  // symbol type
    .domain(data.map(symbolValue))
    .range(symbols);
// Instance of SymbolGenerator to generate the symbols
    const symbolGenerator =  symbol()
```



```
    const marks = data.map((d) => ({
      x: x(xValue(d)),
      y: y(yValue(d)),
      pathD: symbolGenerator.type(symbolScale(symbolValue(d)))(),
    }));

    const circles = selection
      .selectAll('path')
      .data(marks)
      .join('path')
      .attr('d',d => d.pathD)
      .attr('transform', (d) => `translate(${d.x},${d.y})`);
      ;

    selection
      .selectAll('.y-axis')
      .data([null])
      .join('g')
      .attr('class', 'y-axis')
      .attr('transform', `translate(${margin.left},0)`)
      .call(axisLeft(y));

    selection
      .selectAll('.x-axis')
      .data([null])
      .join('g')
      .attr('class', 'x-axis')
      .attr(
        'transform',
        `translate(0,${height - margin.bottom})`
      )
      .call(axisBottom(x));


  };
```



```
  // accessors: they work as setters and getters
  my.width = function (_) {
    return arguments.length ? ((width = +_), my) : width;
  };

  my.height = function (_) {
    return arguments.length ? ((height = +_), my) : height;
  };

  my.data = function (_) {
    return arguments.length ? ((data = _), my) : data;
  };

  my.xValue = function (_) {
    return arguments.length ? ((xValue = _), my) : xValue;
  };

  my.yValue = function (_) {
    return arguments.length ? ((yValue = _), my) : yValue;
  };

   my.symbolValue = function (_) {
    return arguments.length ? ((symbolValue = _), my) : symbolValue;
  };

  my.margin = function (_) {
    return arguments.length ? ((margin = _), my) : margin;
  };

  my.radius = function (_) {
    return arguments.length ? ((radius = +_), my) : radius;
  };

  return my;
};
```

VizHub Link:

https://vizhub.com/nehabaddam/4289b23bb1594ea5ba8f95d1b3305980?edit=files&file=scatterPlot.js

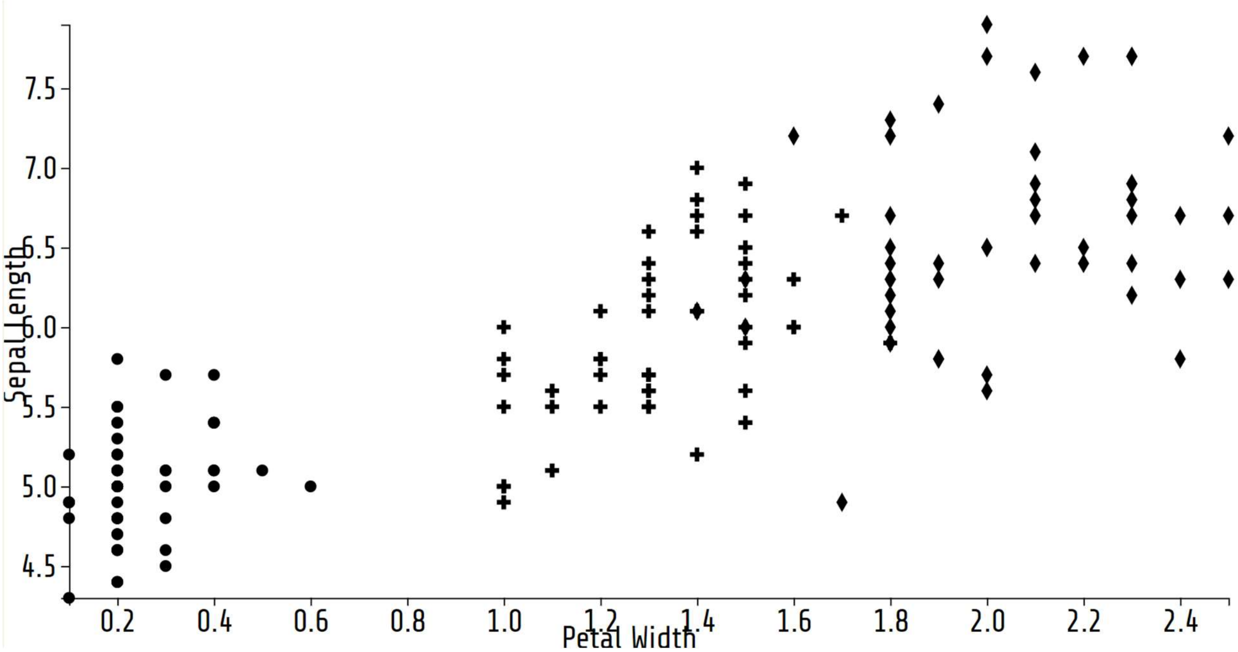
VizHub ID: nehabaddam

Analyzing the chart and describing how different iris species are distributed.

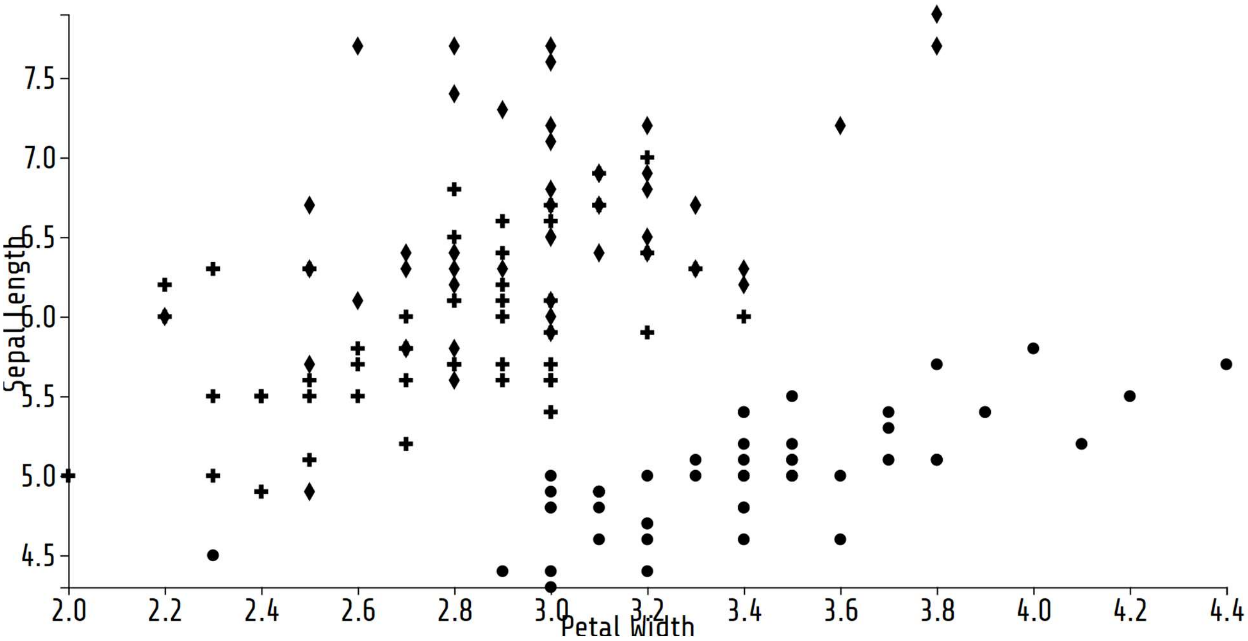
The Iris dataset, initially presented by the statistician Ronald Fisher in 1936, is a popular multivariate dataset. It comprises 150 records of iris flowers with three different species, each with four features, including sepal length, sepal width, petal length, and petal width. It has a record of three species, namely setosa, versicolor, and virginica of 50 each.

The chart being created is a scatter plot visualizing the relationship between the sepal length and petal width of the iris flowers. The iris setosa species is represented using a circle, and the other 2 species are represented using diamond and cross/plus shapes. Each point in the chart represents a single iris flower with its symbol type indicating its species. The chart uses D3.js for data manipulation, scaling, and rendering.

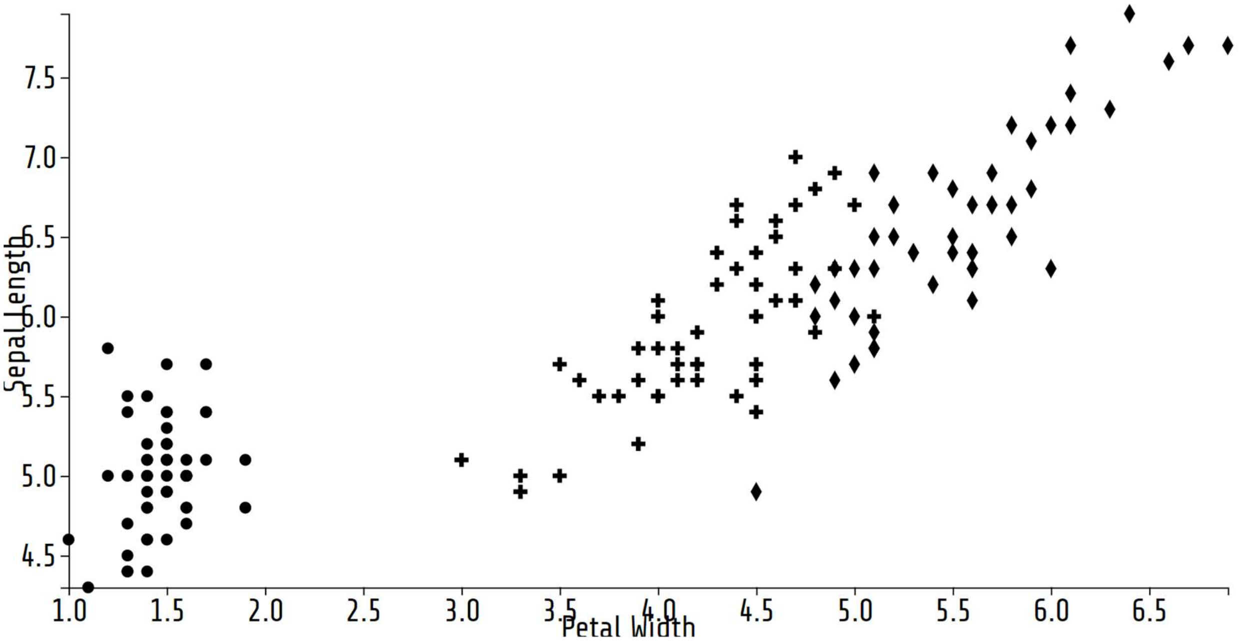
From the chart, we can observe that the iris setosa species can be easily distinguished from the other two species, which appear to overlap in terms of petal width and sepal length.



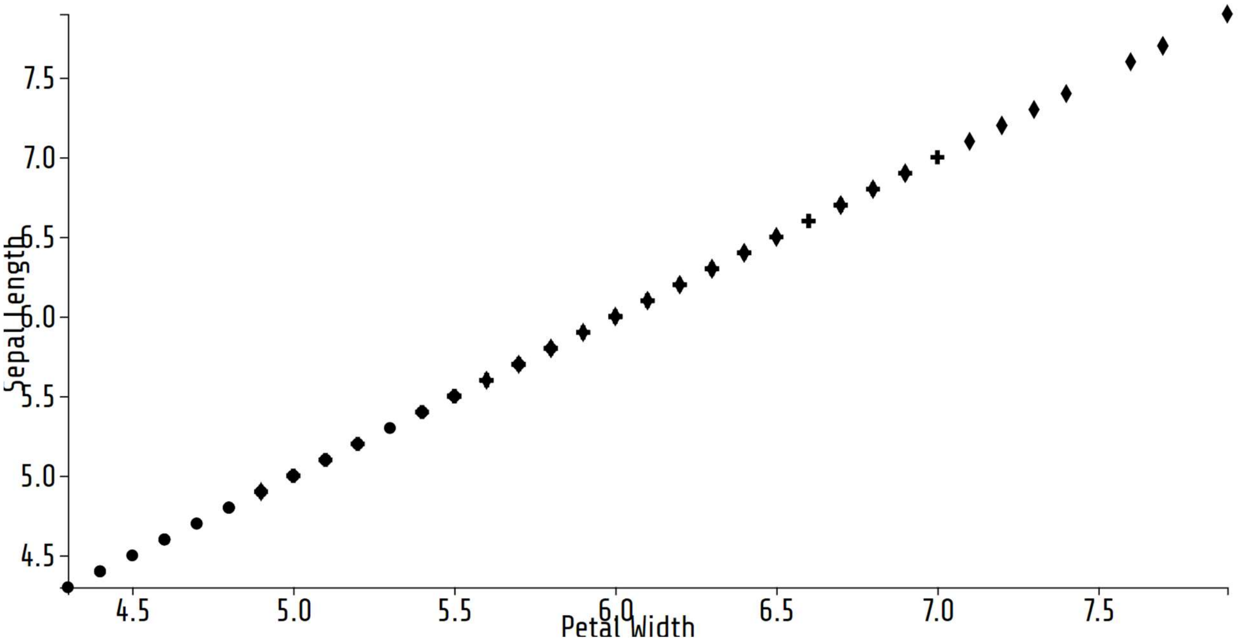
In this chart, all three species are closely aligned, they do not show any differentiation. They are scattered all over the graph.



We can observe that the iris setosa species can be easily distinguished from the other two species, which appear to overlap in terms of petal width and sepal length.



we can observe that the three species are linearly aligned, this shows that with an increase in petal width the sepal length increases.

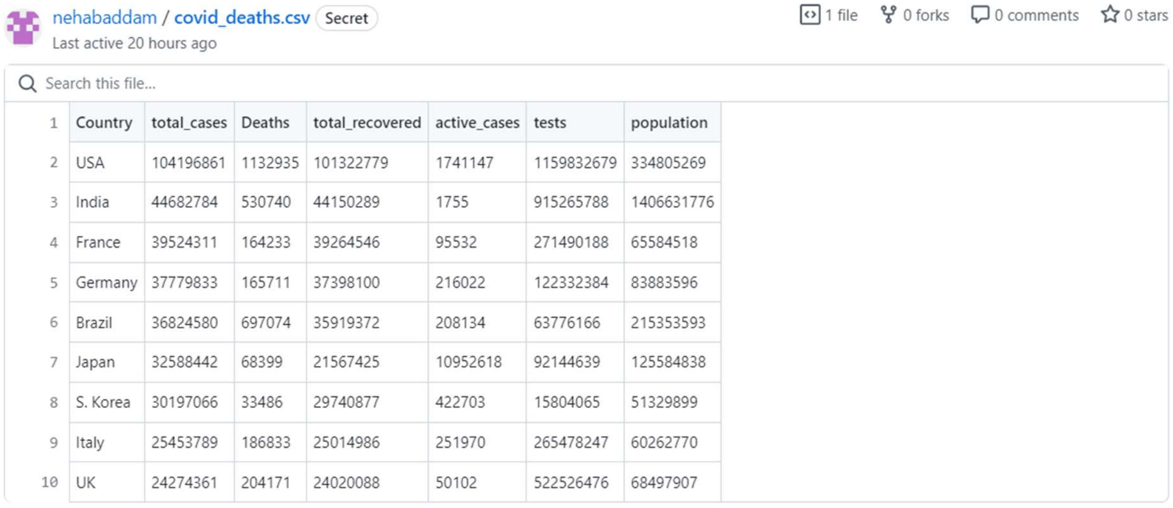


This suggests that these two features may not be sufficient to distinguish between the iris versicolor and virginica species. However, we should note that this is a two-dimensional visualization, and there may be other dimensions that could help differentiate between these species.

Explain the reason why we need to build a reusable scatter plot for the Iris data.

We need to build a reusable scatter plot for the Iris data, it's because the Iris dataset is common and widely used in data visualization and machine learning, and scatter plots are a common way to visualize relationships between two variables. By building a reusable scatter plot function, we can create consistent and standardized visualizations of the Iris data, which can facilitate comparison and analysis across different studies or use cases. Moreover, a reusable function can save time and effort by allowing us to quickly generate scatter plots for the Iris dataset or other similar datasets without having to recreate the entire visualization from scratch every time.

2. I am using the same Covid-19 Dataset that I have used in ICE-6.



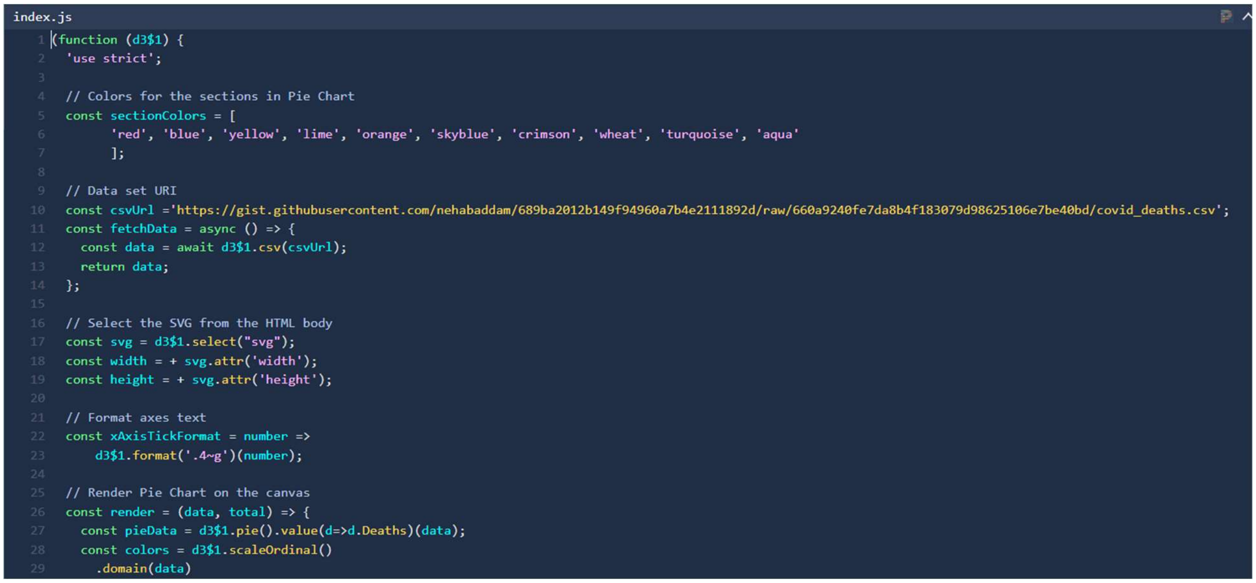
Country	total_cases	Deaths	total_recovered	active_cases	tests	population
USA	104196861	1132935	101322779	1741147	1159832679	334805269
India	44682784	530740	44150289	1755	915265788	1406631776
France	39524311	164233	39264546	95532	271490188	65584518
Germany	37779833	165711	37398100	216022	122332384	83883596
Brazil	36824580	697074	35919372	208134	63776166	215353593
Japan	32588442	68399	21567425	10952618	92144639	125584838
S. Korea	30197066	33486	29740877	422703	15804065	51329899
Italy	25453789	186833	25014986	251970	265478247	60262770
UK	24274361	204171	24020088	50102	522526476	68497907

Creating a Pie Chart.

Now we write a code that creates a pie chart to display COVID-19 death data for different countries using the D3.js library. The data for the chart is obtained from a CSV file located on GitHub. The size of each section in the pie chart corresponds to the number of deaths for that country and is color-coded using predefined colors. The chart is displayed on an SVG element in the HTML body along with text and legend elements.

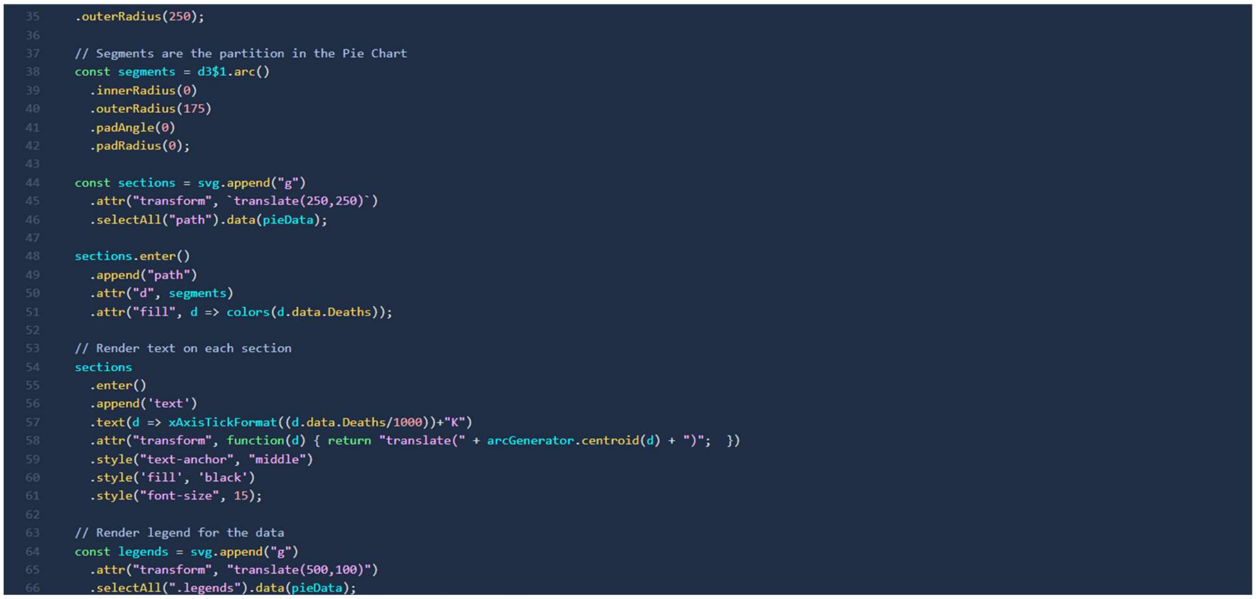
The code defines an array of colors for the chart sections and the URI for the data set. It then uses an async/await function, fetchData, to retrieve the data set from the URI. The SVG element is selected and assigned width and height variables. A xAxisTickFormat function is defined to format the chart's axes text. The render function is then defined to generate the pie chart. It utilizes the d3.pie() function to create pie data from the input data set. The d3.scaleOrdinal() function maps each data point to a color from the predefined array. The arcGenerator object generates arcs for each section in the pie chart, and the segments object partitions each section. The section variable appends path elements for each section in the pie chart, with the fill attribute set to the associated color. The text for each section is generated using the xAxisTickFormat function and positioned using the centroid function of the arcGenerator object. Finally, the legend for the chart is generated using the legends variable, which includes rect and text elements representing the color and value of each section in the pie chart. The run function is defined to fetch the data set using the fetchData function, parse it, and render the pie chart using the render function.

Index.js:

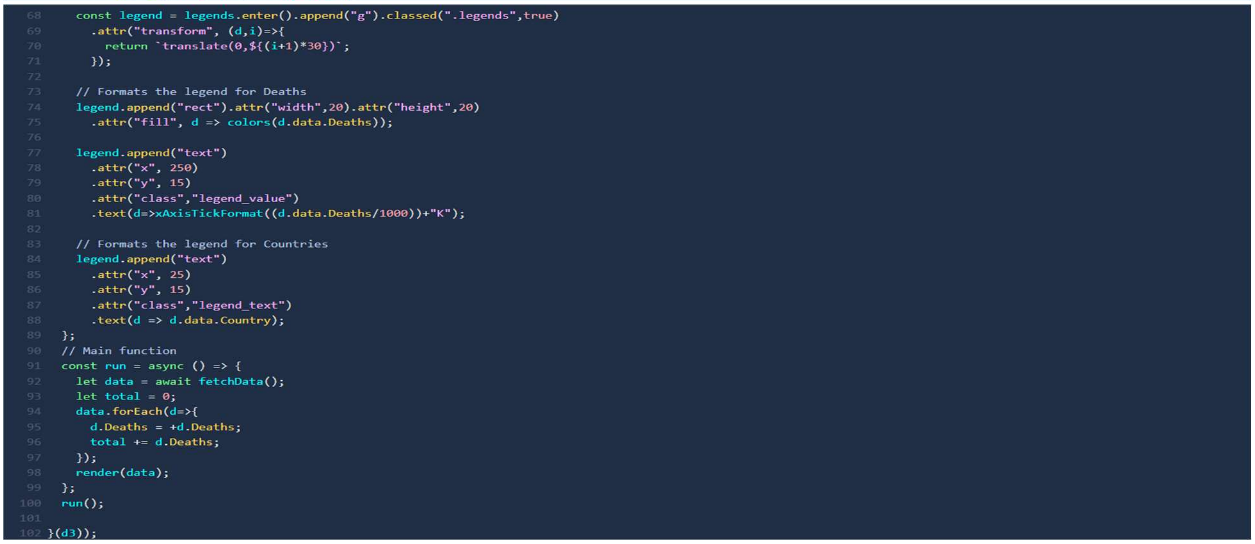


index.js

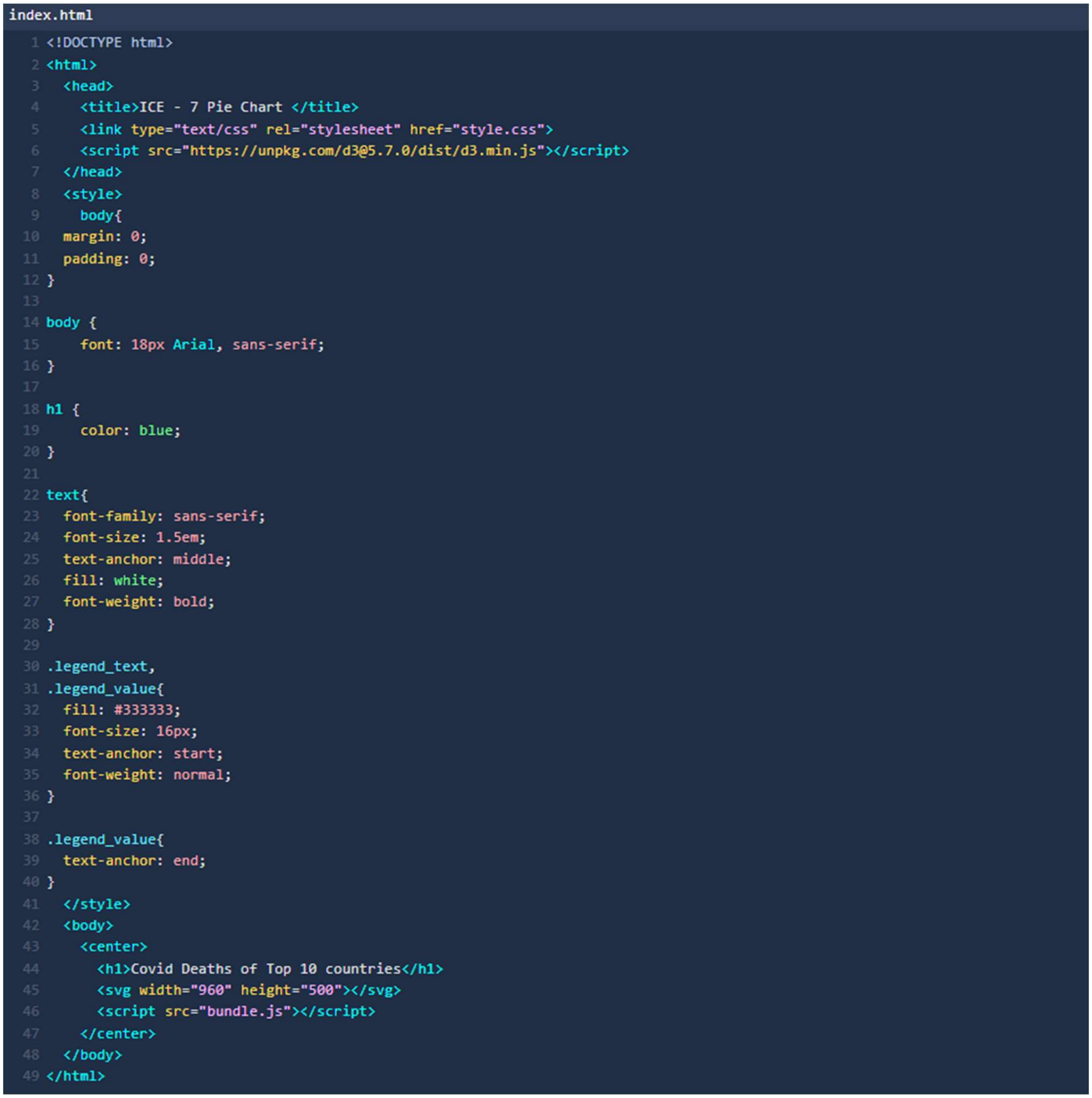
```
(function (d3$1) {
  'use strict';

  // Colors for the sections in Pie Chart
  const sectionColors = [
        'red', 'blue', 'yellow', 'lime', 'orange', 'skyblue', 'crimson', 'wheat', 'turquoise', 'aqua'
        ];

  // Data set URI
  const csvUrl ='https://gist.githubusercontent.com/nehabaddam/689ba2012b149f94960a7b4e2111892d/raw/660a9240fe7da8b4f183079d98625106e7be40bd/covid_deaths.csv';
  const fetchData = async () => {
    const data = await d3$1.csv(csvUrl);
    return data;
  };

  // Select the SVG from the HTML body
  const svg = d3$1.select("svg");
  const width = + svg.attr('width');
  const height = + svg.attr('height');

  // Format axes text
  const xAxisTickFormat = number =>
      d3$1.format('.4~g')(number);

  // Render Pie Chart on the canvas
  const render = (data, total) => {
    const pieData = d3$1.pie().value(d=>d.Deaths)(data);
    const colors = d3$1.scaleOrdinal()
      .domain(data)
```



```
    .outerRadius(250);

    // Segments are the partition in the Pie Chart
    const segments = d3$1.arc()
      .innerRadius(0)
      .outerRadius(175)
      .padAngle(0)
      .padRadius(0);

    const sections = svg.append("g")
      .attr("transform", `translate(250,250)`)
      .selectAll("path").data(pieData);

    sections.enter()
      .append("path")
      .attr("d", segments)
      .attr("fill", d => colors(d.data.Deaths));

    // Render text on each section
    sections
      .enter()
      .append('text')
      .text(d => xAxisTickFormat((d.data.Deaths/1000))+"K")
      .attr("transform", function(d) { return "translate(" + arcGenerator.centroid(d) + ")";  })
      .style("text-anchor", "middle")
      .style('fill', 'black')
      .style("font-size", 15);

    // Render legend for the data
    const legends = svg.append("g")
      .attr("transform", "translate(500,100)")
      .selectAll(".legends").data(pieData);
```



```
    const legend = legends.enter().append("g").classed(".legends",true)
      .attr("transform", (d,i)=>{
        return `translate(0,${(i+1)*30})`;
      });

    // Formats the legend for Deaths
    legend.append("rect").attr("width",20).attr("height",20)
      .attr("fill", d => colors(d.data.Deaths));

    legend.append("text")
      .attr("x", 250)
      .attr("y", 15)
      .attr("class","legend_value")
      .text(d=>xAxisTickFormat((d.data.Deaths/1000))+"K");

    // Formats the legend for Countries
    legend.append("text")
      .attr("x", 25)
      .attr("y", 15)
      .attr("class","legend_text")
      .text(d => d.data.Country);
  };
  // Main function
  const run = async () => {
    let data = await fetchData();
    let total = 0;
    data.forEach(d=>{
      d.Deaths = +d.Deaths;
      total += d.Deaths;
    });
    render(data);
  };
  run();

}(d3));
```



index.html

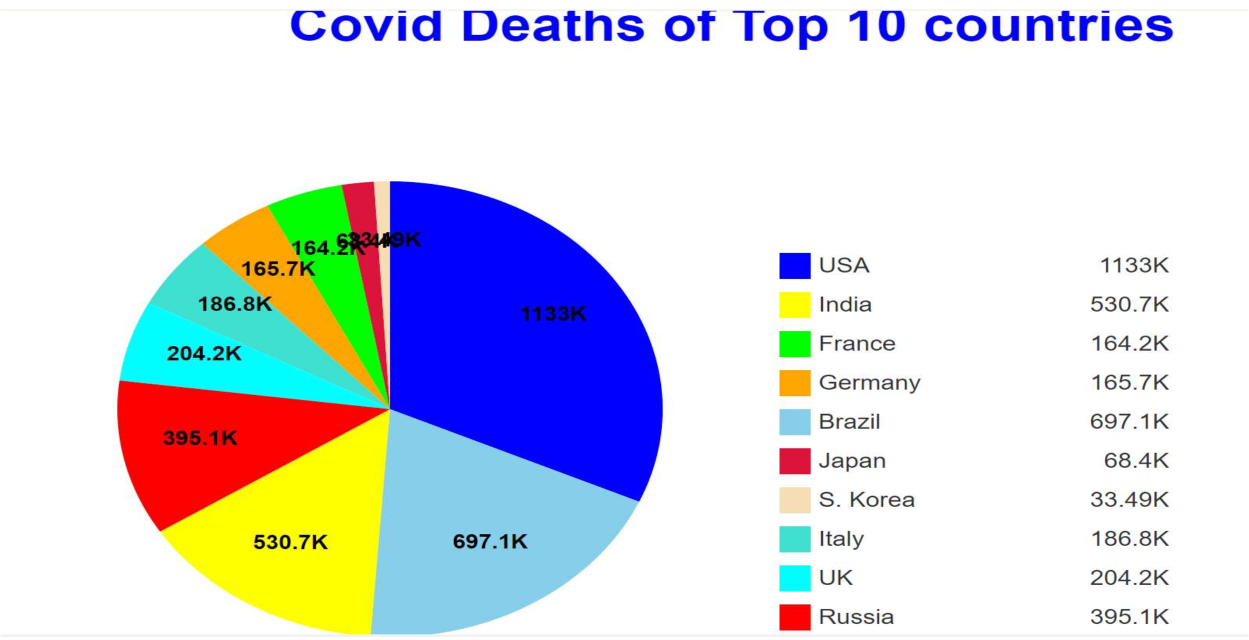
```
<!DOCTYPE html>
<html>
  <head>
    <title>ICE - 7 Pie Chart </title>
    <link type="text/css" rel="stylesheet" href="style.css">
    <script src="https://unpkg.com/d3@5.7.0/dist/d3.min.js"></script>
  </head>
  <style>
    body{
  margin: 0;
  padding: 0;
}

body {
    font: 18px Arial, sans-serif;
}

h1 {
    color: blue;
}

text{
  font-family: sans-serif;
  font-size: 1.5em;
  text-anchor: middle;
  fill: white;
  font-weight: bold;
}

.legend_text,
.legend_value{
  fill: #333333;
  font-size: 16px;
  text-anchor: start;
  font-weight: normal;
}

.legend_value{
  text-anchor: end;
}
  </style>
  <body>
    <center>
      <h1>Covid Deaths of Top 10 countries</h1>
      <svg width="960" height="500"></svg>
      <script src="bundle.js"></script>
    </center>
  </body>
</html>
```

VizHub Link:
https://vizhub.com/nehabaddam/876f40943d7748fd96df4e26136997d7?edit=files&file=index.js

VizHub ID: nehabaddam

Displaying the legend icons next to the chart.

This pie chart displays the number of deaths due to covid in the top 10 countries. Each country is differentiated using a different color. The legends with country names and the death count are shown below.



Analyzing the Pie Chart.

The pie chart visualizes COVID-19 deaths in various countries. Each section in the pie chart represents a country, and the size of each section is proportional to the number of deaths in that country. The sections are color-coded using an array of predefined colors, and a legend is provided to show the color and value of each section. For example, the USA is blue and has 1133k deaths. India is in yellow with 530.7k deaths.

Pros:

Using a pie chart on this dataset has several benefits:

1. Firstly, it provides a simple and visually appealing representation of the distribution of COVID-19 deaths across different countries.
2. The use of colors and legend makes it easy to identify and compare the number of deaths in each country.
3. It is easy for even a layman to understand the chart without needing any extra explanation.

Cons:

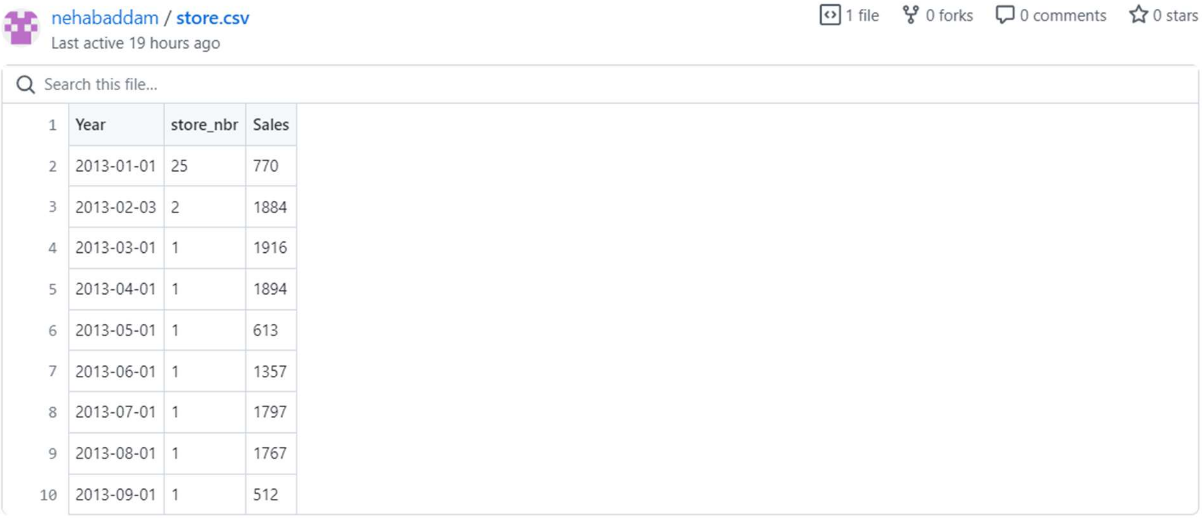
There are also some drawbacks to using a pie chart on this dataset.

1. Firstly, it can be challenging to distinguish between sections that are similar in size.
2. Secondly, it can be challenging to accurately compare the size of sections that are not adjacent to each other in the pie chart.
3. Thirdly, it is hard to show the change in data over time. For example, if we try to show the covid death cases in separate years.
4. Lastly, if there are too many sections or if the sizes of the sections are too small, a pie chart can become cluttered and less informative.

As a result, it is important to consider the advantages and disadvantages of using a pie chart before using it for a particular dataset.

3. I have downloaded the sale data of a store from Kaggle and uploaded it to GitHub Gist. I shall be using that dataset to create a Line Chart.

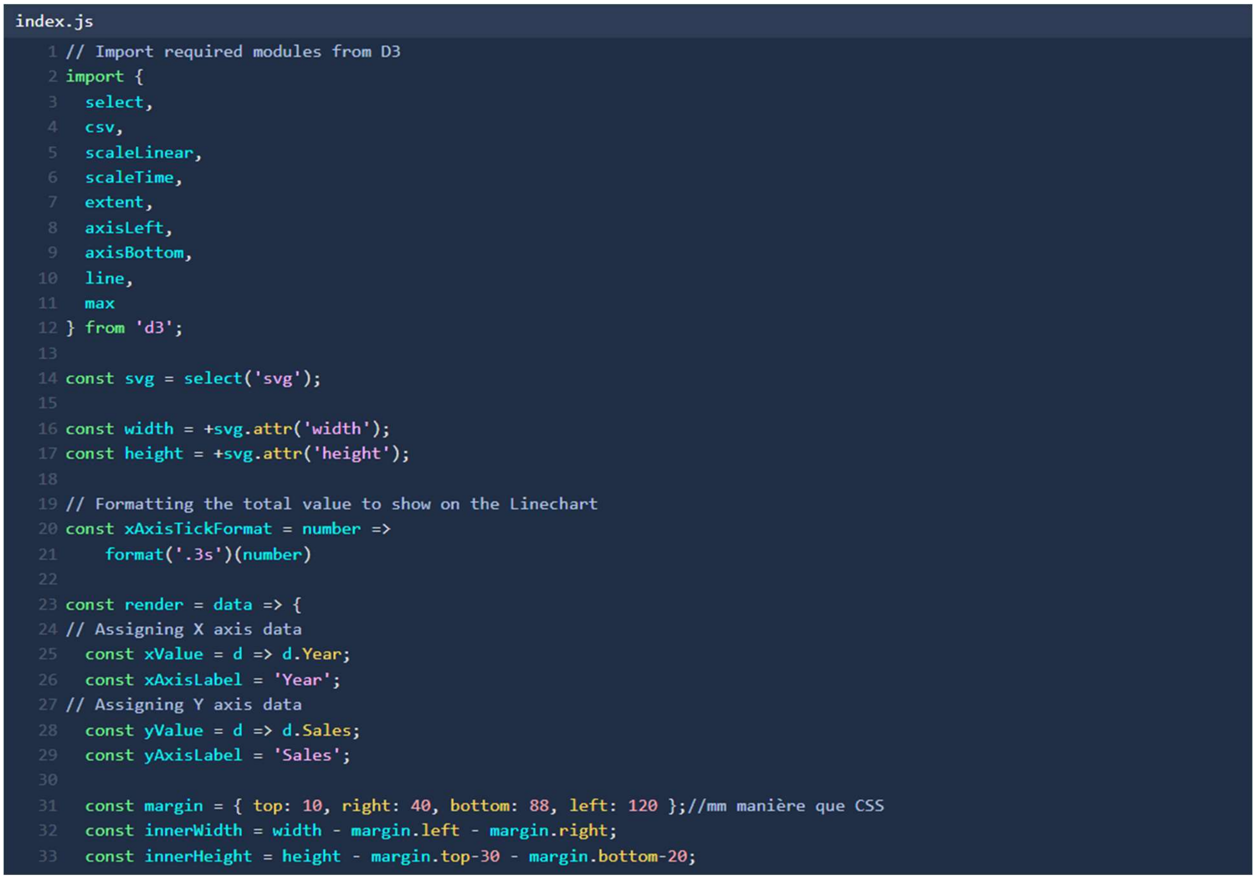
This dataset has 3 columns and used only 57 records from the dataset. It shows the Transaction date, store number, and Sales.



Year	store_nbr	Sales
2013-01-01	25	770
2013-02-03	2	1884
2013-03-01	1	1916
2013-04-01	1	1894
2013-05-01	1	613
2013-06-01	1	1357
2013-07-01	1	1797
2013-08-01	1	1767
2013-09-01	1	512

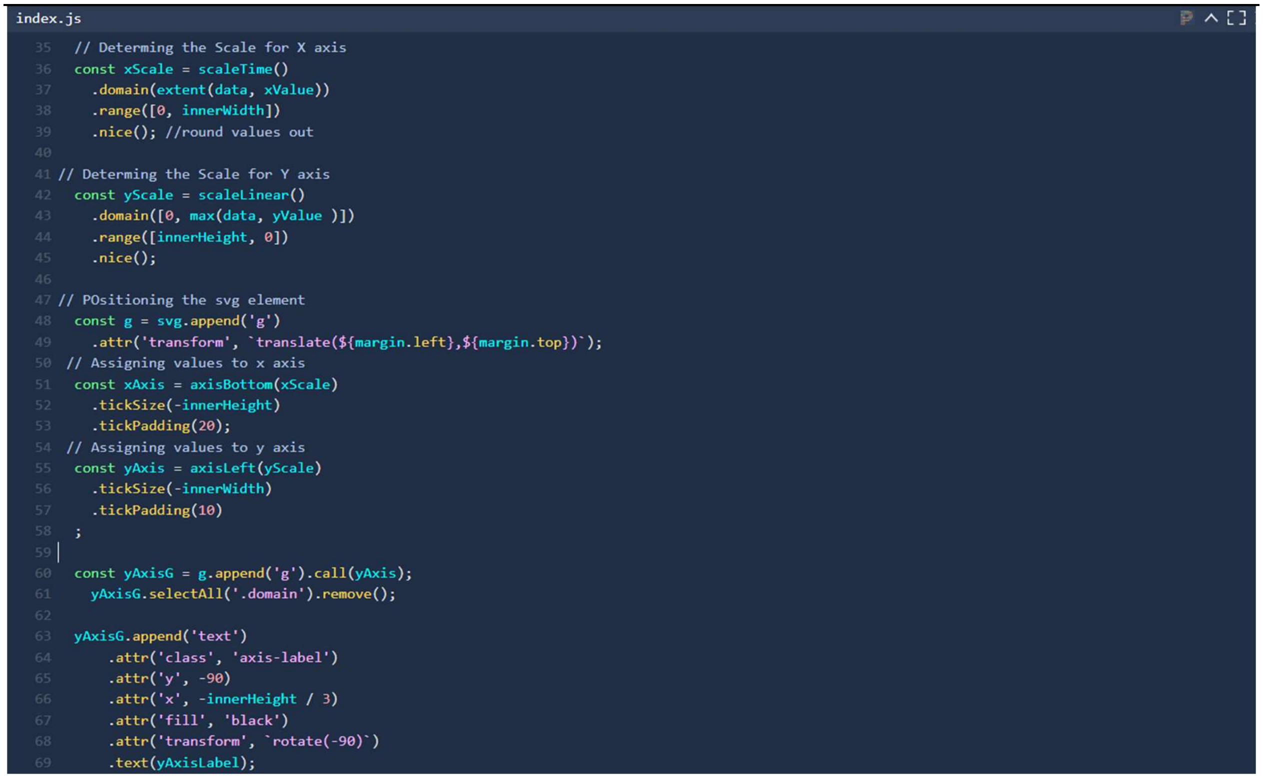
Creating a Line Chart

We are using the D3 library to generate a line chart utilizing CSV data from the GitHub gist. Initially, we import the necessary modules from D3 and select the SVG element where the chart will be rendered. Then we define variables for the width, height, and margin of the chart and define the X and Y value functions for the data using D3's scaleTime and scaleLinear functions, respectively. These scales regulate how the data values will be mapped to the chart's X and Y axes. The X and Y axes are also created using D3's axisBottom and axisLeft functions, respectively. Now we align the chart and add x and y labels. This code generates a line employing D3's line function and appends it to the chart using the path element. This line represents the data's trend over time.



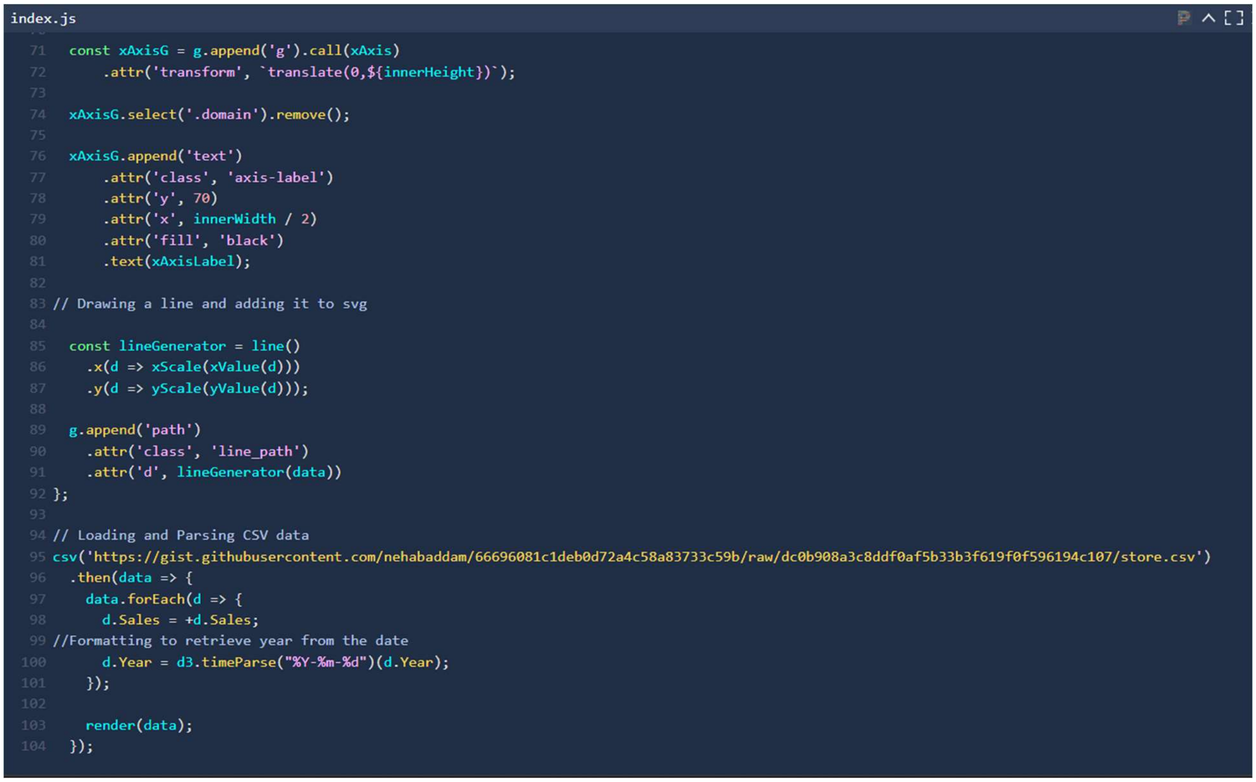
index.js

```
// Import required modules from D3
import {
  select,
  csv,
  scaleLinear,
  scaleTime,
  extent,
  axisLeft,
  axisBottom,
  line,
  max
} from 'd3';

const svg = select('svg');

const width = +svg.attr('width');
const height = +svg.attr('height');

// Formatting the total value to show on the Linechart
const xAxisTickFormat = number =>
    format('.3s')(number)

const render = data => {
// Assigning X axis data
  const xValue = d => d.Year;
  const xAxisLabel = 'Year';
// Assigning Y axis data
  const yValue = d => d.Sales;
  const yAxisLabel = 'Sales';

  const margin = { top: 10, right: 40, bottom: 88, left: 120 };//mm manière que CSS
  const innerWidth = width - margin.left - margin.right;
  const innerHeight = height - margin.top-30 - margin.bottom-20;
```

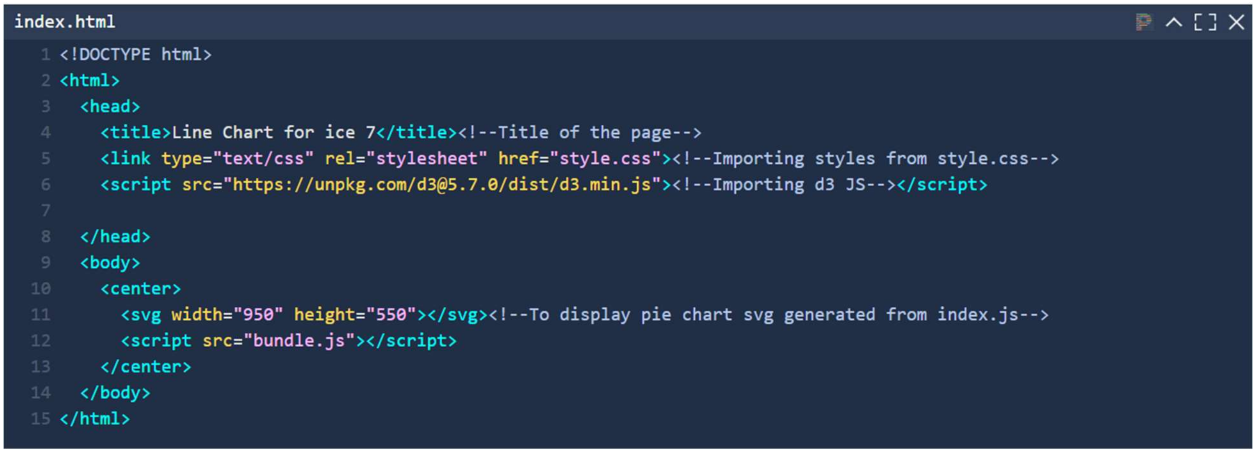


index.js

```
  // Determing the Scale for X axis
  const xScale = scaleTime()
    .domain(extent(data, xValue))
    .range([0, innerWidth])
    .nice(); //round values out

// Determing the Scale for Y axis
  const yScale = scaleLinear()
    .domain([0, max(data, yValue )])
    .range([innerHeight, 0])
    .nice();

// POsitioning the svg element
  const g = svg.append('g')
    .attr('transform', `translate(${margin.left},${margin.top})`);
 // Assigning values to x axis
  const xAxis = axisBottom(xScale)
    .tickSize(-innerHeight)
    .tickPadding(20);
 // Assigning values to y axis
  const yAxis = axisLeft(yScale)
    .tickSize(-innerWidth)
    .tickPadding(10)
  ;

  const yAxisG = g.append('g').call(yAxis);
    yAxisG.selectAll('.domain').remove();

  yAxisG.append('text')
      .attr('class', 'axis-label')
      .attr('y', -90)
      .attr('x', -innerHeight / 3)
      .attr('fill', 'black')
      .attr('transform', `rotate(-90)`)
      .text(yAxisLabel);
```



```javascript
const xAxisG = g.append('g').call(xAxis)
    .attr('transform', `translate(0,${innerHeight})`);

xAxisG.select('.domain').remove();

xAxisG.append('text')
    .attr('class', 'axis-label')
    .attr('y', 70)
    .attr('x', innerWidth / 2)
    .attr('fill', 'black')
    .text(xAxisLabel);

// Drawing a line and adding it to svg

  const lineGenerator = line()
    .x(d => xScale(xValue(d)))
    .y(d => yScale(yValue(d)));

  g.append('path')
    .attr('class', 'line_path')
    .attr('d', lineGenerator(data))
};

// Loading and Parsing CSV data
csv('https://gist.githubusercontent.com/nehabaddam/66696081c1deb0d72a4c58a83733c59b/raw/dc0b908a3c8ddf0af5b33b3f619f0f596194c107/store.csv')
  .then(data => {
    data.forEach(d => {
      d.Sales = +d.Sales;
//Formatting to retrieve year from the date
      d.Year = d3.timeParse("%Y-%m-%d")(d.Year);
    });

    render(data);
  });
```



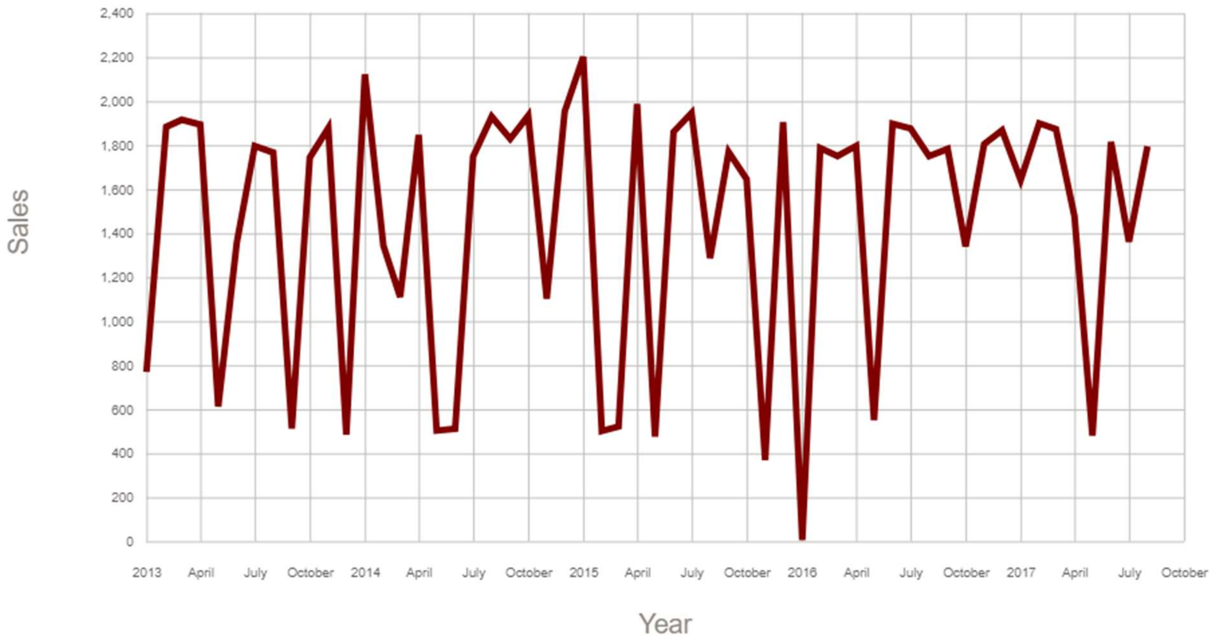
```html
<!DOCTYPE html>
<html>
  <head>
    <title>Line Chart for ice 7</title><!--Title of the page-->
    <link type="text/css" rel="stylesheet" href="style.css"><!--Importing styles from style.css-->
    <script src="https://unpkg.com/d3@5.7.0/dist/d3.min.js"><!--Importing d3 JS--></script>

  </head>
  <body>
    <center>
      <svg width="950" height="550"></svg><!--To display pie chart svg generated from index.js-->
      <script src="bundle.js"></script>
    </center>
  </body>
</html>
```

VizHub Link:
https://vizhub.com/nehabaddam/2f81bc588cfc4744b83258305aeaf171?edit=files&file=index.js

VizHub ID: nehabaddam

Add labels for the x-axis and y-axis for the line chart.

The x and y-axis labels are added. The y-axis shows the Sales data and X-axis shows the Year, it shows the sales data of a store between the years 2013-2017.



Analyze the Line chart.

The code creates a line chart that displays the sales trend of a store over a period of five years. The X-axis of the chart displays the years from 2013 to 2017, while the Y-axis displays the sales amount in dollars. The line on the chart depicts the trend of sales over time, with higher points indicating higher sales and lower points indicating lower sales. For example, the sales in July 2013 is around 1800.

Pros:

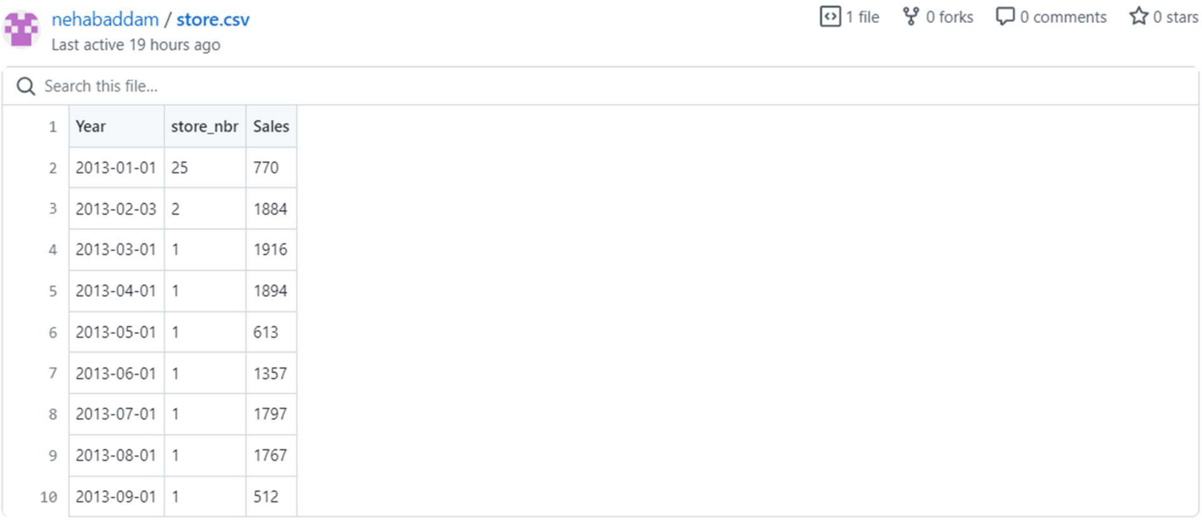
1. It effectively communicates the overall trend of sales over time, and the labeled axes make it easy to understand.
2. The graph makes it easy to understand the maximum and minimums of data. In 2018, the sale is zero, that's the minimum.
3. It is easy to observe changes in data over time. Here over the years 2013-2017.

Cons:

1. However, one disadvantage of this type of chart is that it may not provide detailed information about individual sales transactions or specific events that impact sales.
2. If the dataset includes multiple stores or product lines, it may not be the best choice for comparing their performance.
3. A line chart is an appropriate visualization choice if the goal is to convey the trend of sales over time. However, if a more detailed analysis is required, a different type of visualization or additional data may be necessary.

4. I am using the same dataset as in the third question to create an area chart.

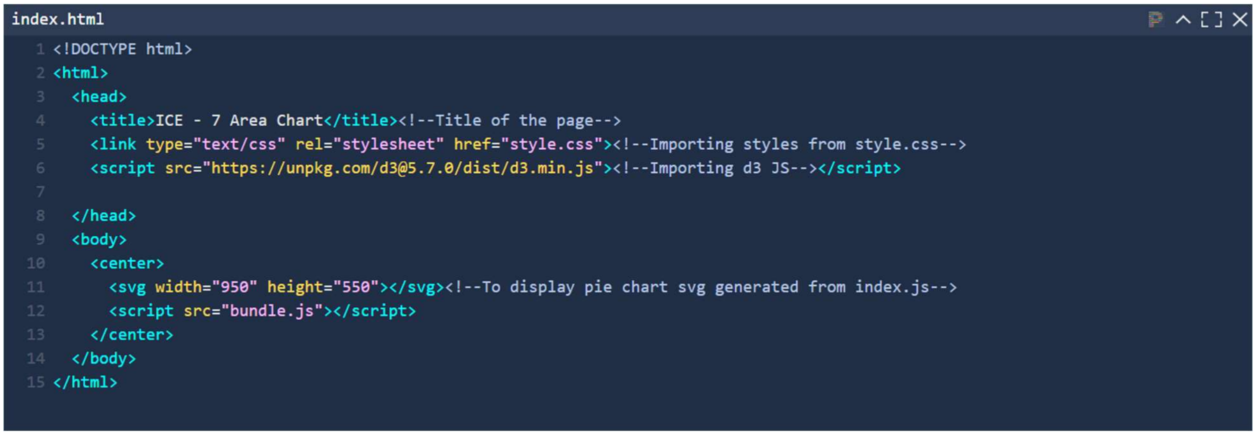
This dataset has 3 columns and used only 57 records from the dataset. It shows the Transaction date, store number, and Sales.



Year	store_nbr	Sales
2013-01-01	25	770
2013-02-03	2	1884
2013-03-01	1	1916
2013-04-01	1	1894
2013-05-01	1	613
2013-06-01	1	1357
2013-07-01	1	1797
2013-08-01	1	1767
2013-09-01	1	512

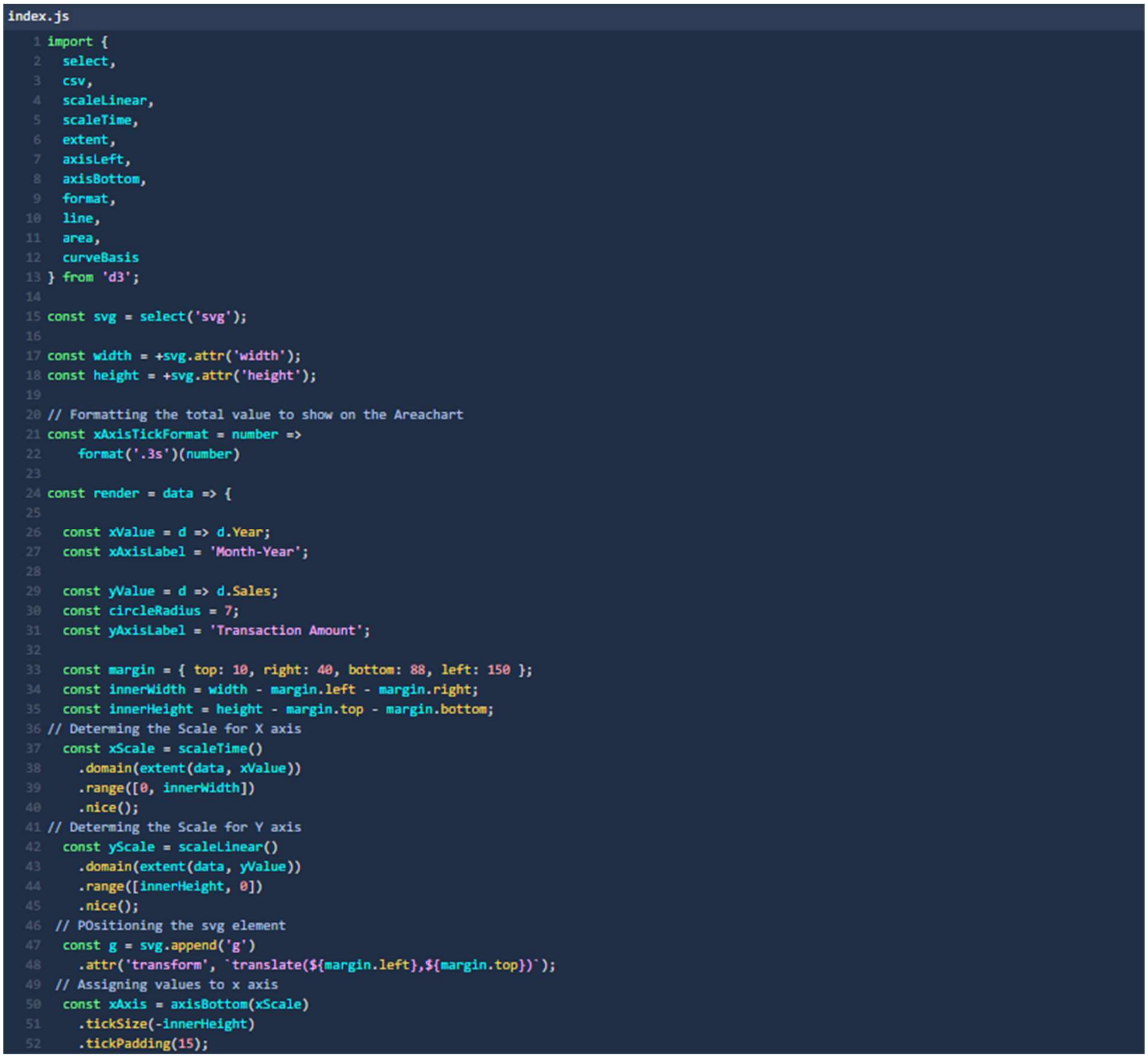
Creating an Area Chart.

The code uses the D3 library to produce an area chart and line chart. Firstly, we load the data from the CSV file in GitHub gist and import the necessary D3 modules, and select the SVG element where the charts will be created. Then we define variables for chart width, height, and margin, and define the X and Y value functions for the data. X and Y scales are created using D3's scaleTime and scaleLinear functions respectively, and X and Y axes are also formed using D3's axisBottom and axisLeft functions. Next, the chart is positioned, and X and Y axes are appended to it with their respective labels. Finally, D3's line and area functions are utilized to generate the line and area charts, which are added to the chart using the path element. The line chart displays the data trend over time, while the area chart represents the area under the line. The code loads and formats CSV data from a GitHub gist URL, where Sales values are converted to numbers, and Year values are transformed to date format. We use CSS to just display the area chart. To compare both area and line chart, we can remove the line-chart function in CSS.



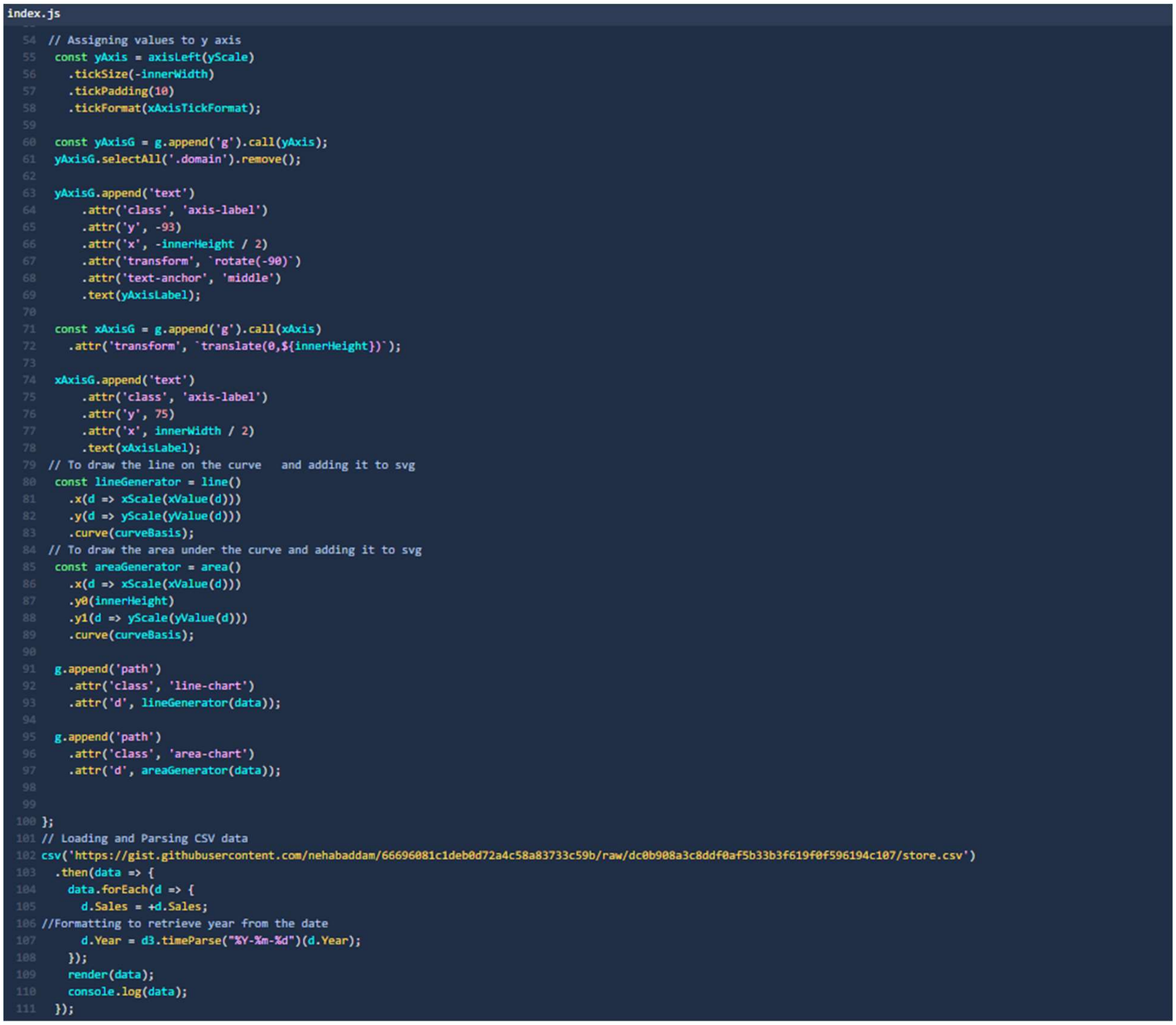
index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>ICE - 7 Area Chart</title><!--Title of the page-->
    <link type="text/css" rel="stylesheet" href="style.css"><!--Importing styles from style.css-->
    <script src="https://unpkg.com/d3@5.7.0/dist/d3.min.js"><!--Importing d3 JS--></script>

  </head>
  <body>
    <center>
      <svg width="950" height="550"></svg><!--To display pie chart svg generated from index.js-->
      <script src="bundle.js"></script>
    </center>
  </body>
</html>
```



index.js

```javascript
import {
  select,
  csv,
  scaleLinear,
  scaleTime,
  extent,
  axisLeft,
  axisBottom,
  format,
  line,
  area,
  curveBasis
} from 'd3';

const svg = select('svg');

const width = +svg.attr('width');
const height = +svg.attr('height');

// Formatting the total value to show on the Areachart
const xAxisTickFormat = number =>
    format('.3s')(number)

const render = data => {

  const xValue = d => d.Year;
  const xAxisLabel = 'Month-Year';

  const yValue = d => d.Sales;
  const circleRadius = 7;
  const yAxisLabel = 'Transaction Amount';

  const margin = { top: 10, right: 40, bottom: 88, left: 150 };
  const innerWidth = width - margin.left - margin.right;
  const innerHeight = height - margin.top - margin.bottom;
// Determing the Scale for X axis
  const xScale = scaleTime()
    .domain(extent(data, xValue))
    .range([0, innerWidth])
    .nice();
// Determing the Scale for Y axis
  const yScale = scaleLinear()
    .domain(extent(data, yValue))
    .range([innerHeight, 0])
    .nice();
 // POsitioning the svg element
  const g = svg.append('g')
    .attr('transform', `translate(${margin.left},${margin.top})`);
 // Assigning values to x axis
  const xAxis = axisBottom(xScale)
    .tickSize(-innerHeight)
    .tickPadding(15);
```



index.js

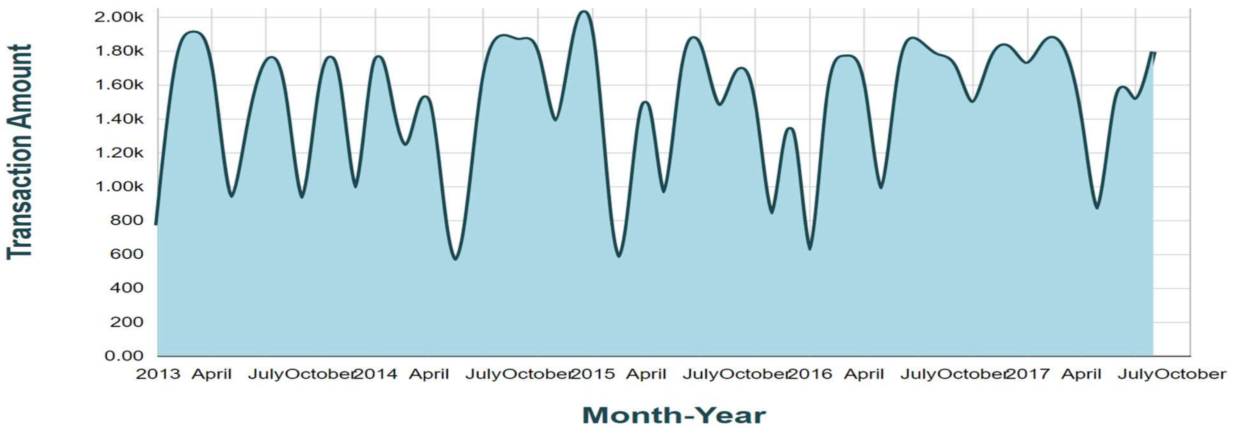
```
 // Assigning values to y axis
  const yAxis = axisLeft(yScale)
    .tickSize(-innerWidth)
    .tickPadding(10)
    .tickFormat(xAxisTickFormat);

  const yAxisG = g.append('g').call(yAxis);
  yAxisG.selectAll('.domain').remove();

  yAxisG.append('text')
      .attr('class', 'axis-label')
      .attr('y', -93)
      .attr('x', -innerHeight / 2)
      .attr('transform', `rotate(-90)`)
      .attr('text-anchor', 'middle')
      .text(yAxisLabel);

  const xAxisG = g.append('g').call(xAxis)
    .attr('transform', `translate(0,${innerHeight})`);

  xAxisG.append('text')
      .attr('class', 'axis-label')
      .attr('y', 75)
      .attr('x', innerWidth / 2)
      .text(xAxisLabel);
 // To draw the line on the curve   and adding it to svg
  const lineGenerator = line()
    .x(d => xScale(xValue(d)))
    .y(d => yScale(yValue(d)))
    .curve(curveBasis);
 // To draw the area under the curve and adding it to svg
  const areaGenerator = area()
    .x(d => xScale(xValue(d)))
    .y0(innerHeight)
    .y1(d => yScale(yValue(d)))
    .curve(curveBasis);

  g.append('path')
    .attr('class', 'line-chart')
    .attr('d', lineGenerator(data));

  g.append('path')
    .attr('class', 'area-chart')
    .attr('d', areaGenerator(data));


};
// Loading and Parsing CSV data
csv('https://gist.githubusercontent.com/nehabaddam/66696081c1deb0d72a4c58a83733c59b/raw/dc0b908a3c8ddf0af5b33b3f619f0f596194c107/store.csv')
  .then(data => {
    data.forEach(d => {
      d.Sales = +d.Sales;
//Formatting to retrieve year from the date
      d.Year = d3.timeParse("%Y-%m-%d")(d.Year);
    });
    render(data);
    console.log(data);
  });
```

VizHub Link:
https://vizhub.com/nehabaddam/0e956813bb464c9b8d57ea0462f9bcc8?edit=files&file=style.css

VizHub ID: nehabaddam

Add labels for the x-axis and y-axis for the area chart.

The x and y-axis labels are added. The y-axis shows the Sales data and X-axis shows the Year, it shows the sales data of a store between the years 2013-2017. For example, the sales in the month of July 2013 are under 1800.



Analyze the area chart.

The chart displays two types of visualizations - a line chart and an area chart. The data represented in the chart shows transaction sales data over time. The X-axis of the chart displays dates, and the Y-axis represents the total transaction amount. The line chart displays the trend of transaction sales data over time, while the area chart represents the total transaction sales for each period.

Differences between the area chart and line chart on this dataset:

1. The line chart provides a clear representation of the ups and downs of the transaction sales data over time, making it easy to spot changes and trends in the data. On the other hand, the area chart displays the total transaction sales for each period, providing an overall view of the magnitude of the sales.
2. The line chart has a narrower range on the Y-axis than the area chart, making it easier to identify changes in the data. The area chart, on the other hand, has a wider range on the Y-axis, making it easier to compare the total transaction sales for each period.
3. Furthermore, the line chart has a more continuous appearance compared to the area chart, which appears more solid. This is because the area chart shows the filled area below the line, while the line chart only displays the line.
4. Both visualizations provide useful insights into the transaction sales data over time, but the line chart is more suitable for analyzing trends and changes, while the area chart is more appropriate for comparing the total transaction sales for each period.