**DATA VISUALIZATION**

**1. Understanding Data Visualization**

* **Definition**: Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.
* **Purpose**: The main goal is to communicate information clearly and efficiently to users. It's used in various fields including business, science, engineering, and social sciences.

**2. Types of Data Visualizations**

* **Bar Charts**: Useful for comparing quantities among different groups.
* **Line Graphs**: Ideal for showing trends over time.
* **Pie Charts**: Good for showing relative proportions or percentages of a whole.
* **Scatter Plots**: Used to show relationships between two variables.
* **Heat Maps**: Great for representing the intensity of data at geographical points.
* **Histograms**: Helpful in showing the distribution of a dataset.

**3. Basic Principles of Effective Data Visualization**

* **Clarity**: The visualization should be easy to understand.
* **Accuracy**: Data should be represented accurately to avoid misleading the audience.
* **Efficiency**: The choice of visualization should convey information in the simplest way possible.
* **Aesthetics**: Visualizations should be visually appealing but not overly complex.

**4. Tools for Data Visualization**

* **Excel/Google Sheets**: Basic, but powerful tools for simple visualizations.
* **Tableau**: A more advanced tool for creating a wide range of interactive and complex visualizations.
* **PowerBI**: Similar to Tableau, often used in business contexts.
* **Python (libraries like Matplotlib, Seaborn, Plotly)**: For those who are comfortable with programming, Python offers extensive libraries for detailed and customized visualizations.
* **R (ggplot2, Shiny)**: Another programming option for statisticians and data scientists.

**HTML FILE**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Lab 5</title>

    <script src="https://d3js.org/d3.v6.min.js"></script>

    <script src="https://d3js.org/topojson.v2.min.js"></script>

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

    <style>

        .bar {

            fill: steelblue;

        }

        .line {

            fill: none;

            stroke: orange;

            stroke-width: 2;

        }

    </style>

</head>

<body>

    <h3>LE PHUOC DAU </h3>

    <h4 class="id"> ITITIU20185 </h4>

    <h4 class="course"> Data science and data visualization </h4>

    <h4 class="labname"> Lab 5 –D3.js - Interaction  <h4>

    <p>This is all my own work. I did not copy the code from any other source</p>

    <svg width="960" height="600"></svg>

    <svg id="combinationChart" width="700" height="400"></svg>

    <script src="test.js" type="text/javascript"></script>

    </body>

**SAMPLE CODE**

A graph of a bar graph

Description automatically generated

const data = [4, 8, 15, 16, 23, 42];

// Set dimensions and margins for the graph

const margin = { top: 20, right: 30, bottom: 40, left: 90 },

      width = 460 - margin.left - margin.right,

      height = 400 - margin.top - margin.bottom;

// Append the svg object to the body of the page

const svg = d3.select("body")

              .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})`);

// X scale and Axis

const x = d3.scaleLinear()

            .domain([0, d3.max(data)])

            .range([0, width]);

svg.append("g")

   .attr("transform", `translate(0,${height})`)

   .call(d3.axisBottom(x));

// Y scale and Axis

const y = d3.scaleBand()

            .range([0, height])

            .domain(data.map((d, i) => i))

            .padding(.1);

svg.append("g")

   .call(d3.axisLeft(y));

// Bars

svg.selectAll("myRect")

   .data(data)

   .enter()

   .append("rect")

   .attr("x", x(0) )

   .attr("y", (d, i) => y(i))

   .attr("width", d => x(d))

   .attr("height", y.bandwidth())

   .attr("class", "bar");

A blue and orange pie chart

Description automatically generated

   // Sample data

   const data = [

    { label: "A", value: 30 },

    { label: "B", value: 20 },

    // ... add more data points here

];

// Width, height and radius of the pie chart

const width = 360;

const height = 360;

const radius = Math.min(width, height) / 2;

// Create primary <g> element

const g = d3.select('#pieChart')

    .attr('width', width)

    .attr('height', height)

    .append('g')

    .attr('transform', 'translate(' + radius + ',' + radius + ')');

// Create the pie chart layout

const pie = d3.pie().value(d => d.value);

// Define the arc for the pie chart

const arc = d3.arc().innerRadius(0).outerRadius(radius);

// Define the colors for the pie chart

const color = d3.scaleOrdinal(d3.schemeCategory10);

// Render the pie chart

const path = g.selectAll('path')

    .data(pie(data))

    .enter()

    .append('path')

    .attr('d', arc)

    .attr('fill', d => color(d.data.label));

A graph with black dots

Description automatically generated

 // Assume we have some data like the following:

 const data = [

    { weight: 2000, mileage: 30 },

    { weight: 2500, mileage: 28 },

    { weight: 3000, mileage: 25 },

    { weight: 3500, mileage: 23 },

    { weight: 4000, mileage: 20 },

    { weight: 4500, mileage: 18 },

    { weight: 3000, mileage: 30 },

    { weight: 3500, mileage: 27 },

    { weight: 400, mileage: 24 },

    { weight: 300, mileage: 30 },

    { weight: 350, mileage: 27 },

    { weight: 4600, mileage: 24 },

    { weight: 3200, mileage: 30 },

    { weight: 3100, mileage: 27 },

    { weight: 4700, mileage: 24 },

    // ... more data

];

// Set dimensions and margins for the graph

const margin = { top: 20, right: 20, bottom: 30, left: 40 },

      width = 720 - margin.left - margin.right,

      height = 432 - margin.top - margin.bottom;

// Append SVG object to the body, set dimensions, and create a group element

const svg = d3.select("svg")

    .attr("width", width + margin.left + margin.right)

    .attr("height", height + margin.top + margin.bottom)

  .append("g")

    .attr("transform", `translate(${margin.left},${margin.top})`);

// Create X scale

const x = d3.scaleLinear()

    .domain([2000, 5000]) // Define domain based on your data

    .range([ 0, width ]);

// Add X axis

svg.append("g")

    .attr("transform", `translate(0,${height})`)

    .call(d3.axisBottom(x));

// Create Y scale

const y = d3.scaleLinear()

    .domain([10, 40]) // Define domain based on your data

    .range([ height, 0]);

// Add Y axis

svg.append("g")

    .call(d3.axisLeft(y));

// Add X axis label

svg.append("text")

    .attr("text-anchor", "end")

    .attr("x", width)

    .attr("y", height + margin.top + 20)

    .text("Weight (lbs.)")

    .attr("class", "axis-label");

// Add Y axis label

svg.append("text")

    .attr("text-anchor", "end")

    .attr("transform", "rotate(-90)")

    .attr("y", -margin.left + 20)

    .attr("x", -margin.top)

    .text("Mileage (mpg)")

    .attr("class", "axis-label");

// Add dots

svg.append('g')

.selectAll("dot")

.data(data)

.join("circle")

  .attr("class", "dot")

  .attr("cx", d => x(d.weight))

  .attr("cy", d => y(d.mileage))

  .attr("r", 5);

-------------------------------------------------------------------------------------------------------------

A graph of different colored lines

Description automatically generated

 // Sample data

 const data = [

    { year: 2017, Bears: 60, Dolphins: 120, Whales: 100 },

    { year: 2018, Bears: 80, Dolphins: 100, Whales: 140 },

    { year: 2019, Bears: 100, Dolphins: 85, Whales: 130 },

    { year: 2020, Bears: 120, Dolphins: 50, Whales: 150 },

    { year: 2021, Bears: 150, Dolphins: 40, Whales: 160 },

    { year: 2022, Bears: 180, Dolphins: 30, Whales: 170 }

];

// Margin convention

const margin = { top: 30, right: 120, bottom: 30, left: 50 },

      width = 600 - margin.left - margin.right,

      height = 315 - margin.top - margin.bottom;

// Append the svg object to the body of the page

const svg = d3.select("svg")

    .attr("width", width + margin.left + margin.right)

    .attr("height", height + margin.top + margin.bottom)

  .append("g")

    .attr("transform", `translate(${margin.left},${margin.top})`);

// Add X axis

const x = d3.scaleLinear()

    .domain(d3.extent(data, d => d.year))

    .range([ 0, width ]);

svg.append("g")

    .attr("transform", `translate(0,${height})`)

    .call(d3.axisBottom(x).tickFormat(d3.format("d")));

// Add Y axis

const y = d3.scaleLinear()

    .domain([0, d3.max(data, d => Math.max(d.Bears, d.Dolphins, d.Whales))])

    .range([ height, 0 ]);

svg.append("g")

    .call(d3.axisLeft(y));

// Color scale

const color = d3.scaleOrdinal()

    .domain(["Bears", "Dolphins", "Whales"])

    .range(["#1f77b4", "#ff7f0e", "#2ca02c"]);

// Draw the line

const drawLine = d3.line()

    .x(d => x(d.year))

    .y(d => y(d.value));

// Lines

["Bears", "Dolphins", "Whales"].forEach((species, i) => {

    svg.append("path")

        .datum(data.map(d => ({ year: d.year, value: d[species] })))

        .attr("fill", "none")

        .attr("stroke", color(species))

        .attr("stroke-width", 1.5)

        .attr("d", drawLine);

});

// Legend

const legend = svg.selectAll(".legend")

    .data(color.domain())

    .enter().append("g")

    .attr("class", "legend")

    .attr("transform", (d, i) => "translate(0," + i \* 20 + ")");

legend.append("rect")

    .attr("x", width + 20)

    .attr("width", 18)

    .attr("height", 18)

    .style("fill", color);

legend.append("text")

    .attr("x", width + 44)

    .attr("y", 9)

    .attr("dy", ".35em")

    .style("text-anchor", "start")

    .text(d => d);

// Title

svg.append("text")

    .attr("x", (width / 2))

    .attr("y", 0 - (margin.top / 2))

    .attr("text-anchor", "middle")

    .style("font-size", "16px")

    .style("text-decoration", "underline")

    .text("Wildlife Population");

-------------------------------------------------------------------------------------------------------------

A map of the united states

Description automatically generated

 // Width and height

 const width = 960;

 const height = 600;

 // Define path generator

 const path = d3.geoPath(); // path generator that will convert GeoJSON to SVG paths

 // Create SVG element and append map to the SVG

 const svg = d3.select("svg");

 // Load in GeoJSON data

 d3.json("https://d3js.org/us-10m.v1.json").then(function(us) {

     // Bind the data to the SVG and create one path per GeoJSON feature

     svg.append("g")

         .selectAll("path")

         .data(topojson.feature(us, us.objects.states).features)

         .enter()

         .append("path")

         .attr("d", path)

         .style("stroke", "#fff")

         .style("stroke-width", "1");

     // Load in your heatmap data here and bind it to the states

     // Since we don't have real heat index data, we'll create a placeholder

     // for demonstration purposes

     const heatData = d3.range(0, 58, 1).map(function(d) {

         return {

             id: d,

             value: Math.floor(Math.random() \* 100) // This is random placeholder data

         };

     });

     // Define a color scale using heatmap values

     const color = d3.scaleQuantize()

         .range(["#ffffcc", "#ffeda0", "#fed976", "#feb24c", "#fd8d3c", "#fc4e2a", "#e31a1c", "#bd0026", "#800026"])

         .domain([0, 100]);

     // Match the data to the US state by ID and apply the heatmap colors

     svg.append("g")

         .selectAll("path")

         .data(topojson.feature(us, us.objects.states).features)

         .style("fill", function(d) {

             const value = heatData.find(h => h.id === d.id)?.value;

             return value ? color(value) : '#ccc';

         });

     // Add the legend - this is a placeholder for demonstration

     // You will need to implement a proper legend that matches your data

     svg.append("g")

         .attr("class", "legendQuant")

         .attr("transform", "translate(20,20)");

     const legend = d3.legendColor()

         .labelFormat(d3.format(".0f"))

         .scale(color)

         .title("Heat Index")

         .titleWidth(100);

     svg.select(".legendQuant")

         .call(legend);

 });

A graph of different colored bars

Description automatically generated

// Data: an array of objects, each representing a group (e.g., Group A, Group B)

var data = [

    {group: "Group A", series1: 12, series2: 19, series3: 8, series4: 21, series5: 17},

    {group: "Group B", series1: 22, series2: 23, series3: 15, series4: 2, series5: 14}

];

// Set dimensions and margins for the graph

var margin = {top: 30, right: 30, bottom: 70, 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("body")

  .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 + ")");

// X axis

var x = d3.scaleBand()

    .range([0, width])

    .domain(data.map(function(d) { return d.group; }))

    .padding(0.2);

svg.append("g")

    .attr("transform", "translate(0," + height + ")")

    .call(d3.axisBottom(x))

// Add Y axis

var y = d3.scaleLinear()

    .domain([0, 30])

    .range([height, 0]);

svg.append("g")

    .call(d3.axisLeft(y));

// Another scale for subgroup position?

var xSubgroup = d3.scaleBand()

    .domain(["series1", "series2", "series3", "series4", "series5"])

    .range([0, x.bandwidth()])

    .padding([0.05])

// color palette = one color per subgroup

var color = d3.scaleOrdinal()

    .domain(["series1", "series2", "series3", "series4", "series5"])

    .range(['#e41a1c','#377eb8','#4daf4a','#984ea3','#ff7f00'])

// Show the bars

svg.append("g")

    .selectAll("g")

    // Enter in data = loop group per group

    .data(data)

    .enter()

    .append("g")

      .attr("transform", function(d) { return "translate(" + x(d.group) + ",0)"; })

    .selectAll("rect")

    .data(function(d) { return ["series1", "series2", "series3", "series4", "series5"].map(function(key) { return {key: key, value: d[key]}; }); })

    .enter().append("rect")

      .attr("x", function(d) { return xSubgroup(d.key); })

      .attr("y", function(d) { return y(d.value); })

      .attr("width", xSubgroup.bandwidth())

      .attr("height", function(d) { return height - y(d.value); })

      .attr("fill", function(d) { return color(d.key); });

A pie chart and a graph

Description automatically generated

 // Sample data

 const data = [

    { district: 'DI', value: 323 },

    { district: 'GO', value: 165 },

    { district: 'LG', value: 119 },

    // ... (rest of your districts and their values)

];

// Define color scale

const colorScale = d3.scaleOrdinal(d3.schemeCategory10);

// Set dimensions for pie chart

const widthPie = 400, heightPie = 400, radius = Math.min(widthPie, heightPie) / 2;

// Set dimensions for bar chart

const widthBar = 400, heightBar = 400, margin = {top: 20, right: 20, bottom: 30, left: 40};

// Pie chart setup

const pie = d3.pie()

    .sort(null)

    .value(d => d.value)(data);

const arc = d3.arc()

    .innerRadius(0)

    .outerRadius(radius);

const pieSvg = d3.select("#pie")

    .attr("width", widthPie)

    .attr("height", heightPie)

  .append("g")

    .attr("transform", `translate(${widthPie / 2}, ${heightPie / 2})`);

// Bar chart setup

const x = d3.scaleBand()

    .range([0, widthBar - margin.left - margin.right])

    .padding(0.1)

    .domain(data.map(d => d.district));

const y = d3.scaleLinear()

    .range([heightBar - margin.top - margin.bottom, 0])

    .domain([0, d3.max(data, d => d.value)]);

const barSvg = d3.select("#bar")

    .attr("width", widthBar)

    .attr("height", heightBar)

  .append("g")

    .attr("transform", `translate(${margin.left}, ${margin.top})`);

// Draw pie chart

pieSvg.selectAll(".arc")

    .data(pie)

    .enter().append("path")

    .attr("class", "arc")

    .attr("d", arc)

    .attr("fill", (d, i) => colorScale(i)); // Use the color scale for fill

pieSvg.selectAll(".arc text")

    .data(pie)

    .enter().append("text")

    .attr("transform", d => `translate(${arc.centroid(d)})`)

    .attr("dy", "0.35em")

    .text(d => d.data.district);

// Draw bar chart

barSvg.selectAll(".bar")

    .data(data)

    .enter().append("rect")

    .attr("class", "bar")

    .attr("x", d => x(d.district))

    .attr("y", d => y(d.value))

    .attr("width", x.bandwidth())

    .attr("height", d => heightBar - margin.top - margin.bottom - y(d.value))

    .attr("fill", (d, i) => colorScale(i)); // Use the color scale for fill

barSvg.append("g")

    .attr("transform", `translate(0, ${heightBar - margin.top - margin.bottom})`)

    .call(d3.axisBottom(x));

barSvg.append("g")

    .call(d3.axisLeft(y));

A graph with a line and a line

Description automatically generated with medium confidence

// Sample data

const data = [

    { month: 'Tháng 1', revenue: 2000000, ads: 31001000 },

    { month: 'Tháng 2', revenue: 30000, ads: 41000100 },

    { month: 'Tháng 3', revenue: 2000000, ads: 201110000 },

    { month: 'Tháng 4', revenue: 3200000, ads: 501100100 },

    { month: 'Tháng 5', revenue: 2000000, ads: 300000 },

    { month: 'Tháng 6', revenue: 3000000, ads: 4008000 },

    { month: 'Tháng 7', revenue: 2000000, ads: 300000 },

    { month: 'Tháng 8', revenue: 3001000, ads: 431180000 },

    { month: 'Tháng 9', revenue: 2000000, ads: 3008000 },

    { month: 'Tháng 10', revenue: 3000000, ads: 40011800 },

    { month: 'Tháng 11', revenue: 3000000, ads: 3888120 },

    { month: 'Tháng 12', revenue: 3000000, ads: 38475863},

    // ... more months with their respective 'revenue' and 'ads' values

];

// Dimensions and margins

const margin = {top: 20, right: 30, bottom: 40, left: 50},

      width = 700 - margin.left - margin.right,

      height = 400 - margin.top - margin.bottom;

// Append SVG object to the body

const svg = d3.select("#combinationChart")

    .attr("width", width + margin.left + margin.right)

    .attr("height", height + margin.top + margin.bottom)

    .append("g")

    .attr("transform", `translate(${margin.left},${margin.top})`);

// X scale

const x = d3.scaleBand()

    .range([0, width])

    .domain(data.map(d => d.month))

    .padding(0.1);

// Add Y axis for revenue

const yRevenue = d3.scaleLinear()

    .domain([0, d3.max(data, d => d.revenue)])

    .range([height, 0]);

// Y axis for ads (assuming it's a different scale)

const yAds = d3.scaleLinear()

    .domain([0, d3.max(data, d => d.ads)])

    .range([height, 0]);

// Bars for the revenue

svg.selectAll(".bar")

    .data(data)

    .join("rect")

    .attr("class", "bar")

    .attr("x", d => x(d.month))

    .attr("y", d => yRevenue(d.revenue))

    .attr("width", x.bandwidth())

    .attr("height", d => height - yRevenue(d.revenue))

    .attr("fill", "steelblue"); // Fixed color for the bars

// Line for the ads

const line = d3.line()

    .x(d => x(d.month) + x.bandwidth() / 2) // Center the line in the band

    .y(d => yAds(d.ads));

// Draw the line

svg.append("path")

    .datum(data) // 10. Binds data to the line

    .attr("class", "line") // Assign a class for styling

    .attr("d", line) // 11. Calls the line generator

    .attr("stroke", "orange"); // Fixed color for the line

// Add the X Axis

svg.append("g")

    .attr("transform", `translate(0,${height})`)

    .call(d3.axisBottom(x));

// Add the Y Axis for revenue

svg.append("g")

    .call(d3.axisLeft(yRevenue));

// Add the Y Axis for ads

svg.append("g")

    .attr("transform", `translate(${width},0)`)

    .call(d3.axisRight(yAds));