Task 6:Data Visualization for Your Profile Website

Objective:

Implement a skills or stats chart for a profile website using JavaScript and a charting library, bind the chart data to the user's profile information, and make the chart interactive.

Pre-requisites:

- Basic understanding of HTML, CSS, and JavaScript
- Familiarity with a code editor like Visual Studio Code
- Basic understanding of using JavaScript libraries

Concepts Covered:

- Implementing a Skills or Stats Chart
- Dynamic Data Binding
- Interactive Data Display

Concepts:

1. **Implementing a Skills or Stats Chart:**

   Use JavaScript to dynamically create a chart or graph representing skills or statistics. Consider using a simple JavaScript library like Chart.js for this purpose.

```html
<canvas id="skillsChart" width="400" height="400"></canvas>
```

```html
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
    const ctx = document.getElementById('skillsChart').getContext('2d');
    const skillsChart = new Chart(ctx, {
        type: 'bar',
        data: {
            labels: ['HTML', 'CSS', 'JavaScript', 'React'],
            datasets: [{
                label: 'Skill Level',
                data: [85, 75, 90, 80],
                backgroundColor: [
                    'rgba(255, 99, 132, 0.2)',
                    'rgba(54, 162, 235, 0.2)',
                    'rgba(255, 206, 86, 0.2)',
                    'rgba(75, 192, 192, 0.2)'
                ],
                borderColor: [
                    'rgba(255, 99, 132, 1)',
                    'rgba(54, 162, 235, 1)',
                    'rgba(255, 206, 86, 1)',
                    'rgba(75, 192, 192, 1)'
                ],
                borderWidth: 1
            }]
        },
        options: {
            scales: {
                y: {
                    beginAtZero: true
                }
            }
        }
    });
</script>
```

2. **Dynamic Data Binding:**

   Bind the chart data to a user's profile information. Ensure the chart updates when the profile information changes.

```
const profile = {
    name: "John Doe",
    skills: {
        HTML: 85,
        CSS: 75,
        JavaScript: 90,
        React: 80
    }
};

function updateChart(profile) {
    skillsChart.data.labels = Object.keys(profile.skills);
    skillsChart.data.datasets[0].data = Object.values(profile.skills);
    skillsChart.update();
}

// Initial chart update
updateChart(profile);
```

3. **Interactive Data Display:**

   Make the chart interactive (e.g., hover effects to show more details). Provide a way for users to interact with the data, like filtering or sorting.

```
<button onclick="filterSkills('high')">Show High Skills</button>
<button onclick="filterSkills('low')">Show Low Skills</button>
```

```
function filterSkills(level) {
    let filteredSkills = {};
    if (level === 'high') {
        filteredSkills =
Object.fromEntries(Object.entries(profile.skills).filter(([skill, value]) =>
value > 80));
    } else if (level === 'low') {
        filteredSkills =
Object.fromEntries(Object.entries(profile.skills).filter(([skill, value]) =>
value <= 80));
    }
    updateChart({ skills: filteredSkills });
}
```

Setup:

1. **Install Visual Studio Code (VS Code):**

   Download and install VS Code from Visual Studio Code.

2. **Web Browsers:**

   Use Google Chrome or Mozilla Firefox for viewing your webpage and utilizing their developer tools for debugging.

Tasks:

1. **Implement a Skills or Stats Chart (10 minutes):**
   - Use JavaScript to dynamically create a chart or graph representing skills or statistics.
   - Consider using a simple JavaScript library like Chart.js for this purpose.
   - Example:

```
<canvas id="skillsChart" width="400" height="400"></canvas>
```

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
    const ctx = document.getElementById('skillsChart').getContext('2d');
    const skillsChart = new Chart(ctx, {
        type: 'bar',
        data: {
            labels: ['HTML', 'CSS', 'JavaScript', 'React'],
            datasets: [{
                label: 'Skill Level',
                data: [85, 75, 90, 80],
                backgroundColor: [
                    'rgba(255, 99, 132, 0.2)',
                    'rgba(54, 162, 235, 0.2)',
                    'rgba(255, 206, 86, 0.2)',
                    'rgba(75, 192, 192, 0.2)'
                ],
                borderColor: [
                    'rgba(255, 99, 132, 1)',
                    'rgba(54, 162, 235, 1)',
                    'rgba(255, 206, 86, 1)',
                    'rgba(75, 192, 192, 1)'
                ],
                borderWidth: 1
            }]
        },
        options: {
            scales: {
                y: {
                    beginAtZero: true
                }
            }
        }
    });
</script>
```

2. **Dynamic Data Binding (10 minutes):**
    - Bind the chart data to a user's profile information.
    - Ensure the chart updates when the profile information changes.
    - Example:

```
const profile = {
    name: "John Doe",
    skills: {
        HTML: 85,
        CSS: 75,
        JavaScript: 90,
        React: 80
    }
};

function updateChart(profile) {
    skillsChart.data.labels = Object.keys(profile.skills);
    skillsChart.data.datasets[0].data = Object.values(profile.skills);
    skillsChart.update();
}

// Initial chart update
updateChart(profile);
```

3. **Interactive Data Display (10 minutes):**
    - Make the chart interactive (e.g., hover effects to show more details).
    - Provide a way for users to interact with the data, like filtering or sorting.
    - Example:

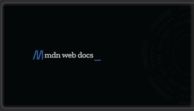
```html
<button onclick="filterSkills('high')">Show High Skills</button>
<button onclick="filterSkills('low')">Show Low Skills</button>
```

```javascript
function filterSkills(level) {
    let filteredSkills = {};
    if (level === 'high') {
        filteredSkills =
Object.fromEntries(Object.entries(profile.skills).filter(([skill, value]) =>
value > 80));
    } else if (level === 'low') {
        filteredSkills =
Object.fromEntries(Object.entries(profile.skills).filter(([skill, value]) =>
value <= 80));
    }
    updateChart({ skills: filteredSkills });
}
```

Instructions:

1. Write the required code in `index.html` and `script.js`.
2. Open the `index.html` file in your web browser to ensure the code displays correctly.
3. Use the browser's developer tools to debug and inspect the elements.

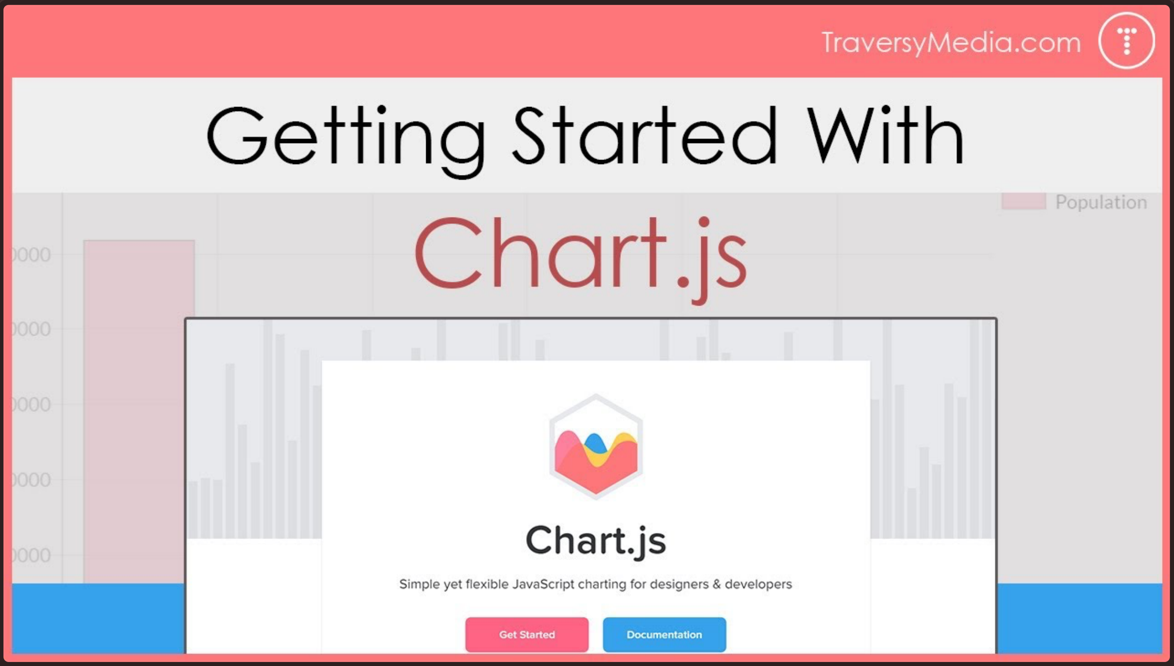
Resources:

- Chart.js | Chart.js
  Open source HTML5 Charts for your website
  https://www.chartjs.org/docs/latest/
- 

  Introduction to the DOM - Web APIs | MDN
  The Document Object Model (DOM) is the data representation of the objects that compri...
  https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
-

  Documentation for Visual Studio Code
  Find out how to set-up and get the most from Visual Studio Code. Optimized for building and d...
  https://code.visualstudio.com/docs

Videos:

- 

Traversy Media 19:26min 501,920 Views 9,425 Likes

GitHub Instructions:

1. **Open in Visual Studio Code:**
   After clicking on the "Open in Visual Studio Code" button from the GitHub Classroom confirmation page, VSCode will open the repository directly. If prompted, select "Open" or "Allow" to open the repository in VSCode.
2. **Open the Terminal in VSCode:**
   In VSCode, open a terminal by selecting Terminal > New Terminal from the top menu.
3. **Complete the Task:**
   In VSCode, write your solution in the `index.html` and `script.js` files.
4. **Run and Test Your Code:**
   Open your `index.html` file in a web browser to ensure it works correctly. Use the following command:

   ```
   open index.html
   ```

5. **Commit Your Changes:**
   In the VSCode terminal, add your changes to git:

   ```
   git add index.html script.js
   ```

   Commit your changes with a meaningful message:

```
git commit -m "Completed task 18"
```

6. **Push Your Changes to Your Repository:**
   Push your changes to your forked repository:

   ```
   git push origin main
   ```

7. **Create a Pull Request:**
   Go to your repository on GitHub.
   Click on the "Pull Requests" tab.
   Click the "New Pull Request" button.
   Ensure the base repository is the original template repository and the base branch is `main` .
   Ensure the head repository is your forked repository and the compare branch is `main` .
   Click "Create Pull Request".
   Add a title and description for your pull request and submit it.

Summary of Commands:

```
# Open in Visual Studio Code

# Open terminal in VSCode

# Complete the task by editing index.html and script.js

# Navigate to the directory containing index.html
cd path/to/your/index.html

# Run your code
open index.html

# Add, commit, and push your changes
git add index.html script.js
git commit -m "Completed task 7"
git push origin main

# Create a pull request on GitHub
```