Sherwin Duran

Task 7-8 Report

1. I began by loading the Iris dataset from the iris.json file using the Fetch API. The result was an array of 150 objects, each representing a flower with properties such as sepalLength, sepalWidth, petalLength, petalWidth, and species.

```
const response = await fetch('data/iris.json');

const irisData = await response.json();
```

2. Using map(), I created a new array called irisesWithColors. Each object in this array includes all original properties plus a new property called color, which is randomly selected from a predefined set of color codes (["#5d3fd3", "#a73fd3", "#d33fb5", "#d35d3f", "#d3a73f"]).

```
const possibleColor = ["#5d3fd3", "#a73fd3", "#d33fb5", "#d35d3f", "#d3a73f"];

const irisesWithColors = irisData.map(iris => ({

 ...iris,

 color: possibleColor[Math.floor(Math.random() * possibleColor.length)]

}));
```

3. Then, using filter(), I generated a new array called filteredIrises by filtering out any flower whose sepalWidth is greater than or equal to 4. This step helps refine the data set based on a specific condition.

```
const filteredIrises = irisesWithColors.filter(iris => iris.sepalWidth < 4);
```

4. I applied reduce() to calculate the average petalLength of all flowers. First, I used reduce to sum up the petalLength values, and then divided that total by the number of objects in the array to get the average.

```
const totalPetalLength = irisesWithColors.reduce((sum, iris) => sum + iris.petalLength, 0);

const avgPetalLength = totalPetalLength / irisesWithColors.length;
```

5. Using find(), I located the first iris object in the dataset that had a petalWidth greater than 1.0. This operation stops at the first match and returns the corresponding flower object.

const foundIris = irisesWithColors.find(iris => iris.petalWidth > 1.0);

6. I used some() to check if at least one flower had a petalLength greater than 10. The result was false, which is expected given the natural size range in the dataset.

const hasPetalLongerThan10 = irisesWithColors.some(iris => iris.petalLength > 10);

7. Another use of some() checked whether there is any flower with a petalLength equal to exactly 4.2. This returned true, showing that at least one such object exists in the array.

const hasPetalLength42 = irisesWithColors.some(iris => iris.petalLength === 4.2);

8. With every(), I tested if all iris objects had a petalWidth less than 3. This returned true, confirming the consistency of this feature across the dataset.

const allPetalWidthUnder3 = irisesWithColors.every(iris => iris.petalWidth < 3);

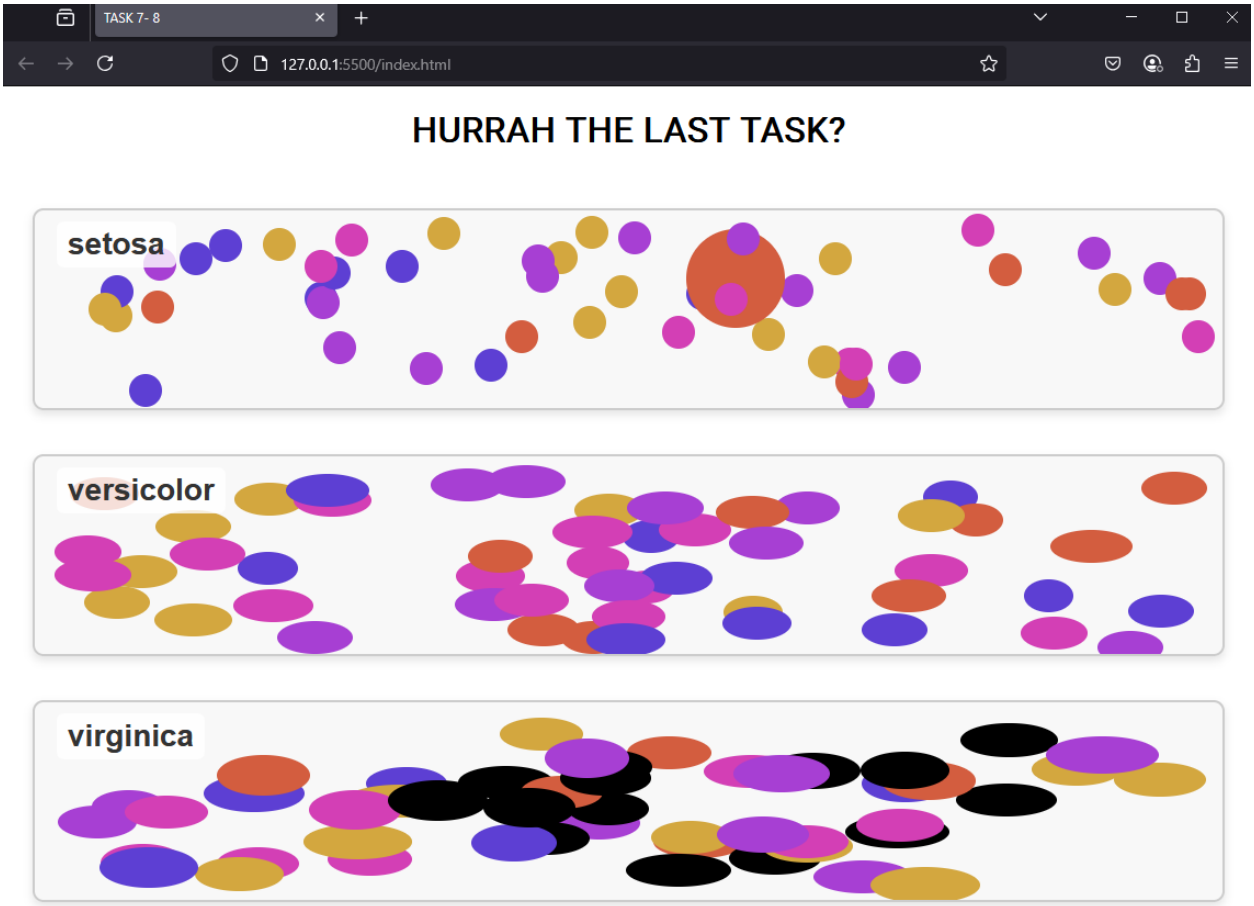
9. Another every() test checked if all iris objects had a sepalWidth greater than 1.2. This also returned true, indicating that no data point falls below this threshold.

const allSepalWidthAbove1_2 = irisesWithColors.every(iris => iris.sepalWidth > 1.2);

10. I used toSorted() to create a sorted version of the irisesWithColors array, named irisesWithColorsSorted. The sorting was done based on petalWidth, from smallest to largest, which helps structure the data better for visual display.

const irisesWithColorsSorted = irisesWithColors.toSorted((a, b) => a.petalWidth - b.petalWidth);

11. In my visualization, I organized the three types of iris flowers into separate rows.



- For **Setosa**, the petals grow bigger when you hover over them.
- The **Versicolor** petals move away from the mouse when you hover over them, making them look like they're avoiding you.
- For **Virginica**, the petals turn black when you hover over them.

I think that these simple interactions can help show the differences between each species in a fun way.