# Lecture Exercise 7 (5/16)

## Submit your team number

**Question** *Submitted May 16th 2023 at 4:44:51 pm*

Please enter your team number.

12

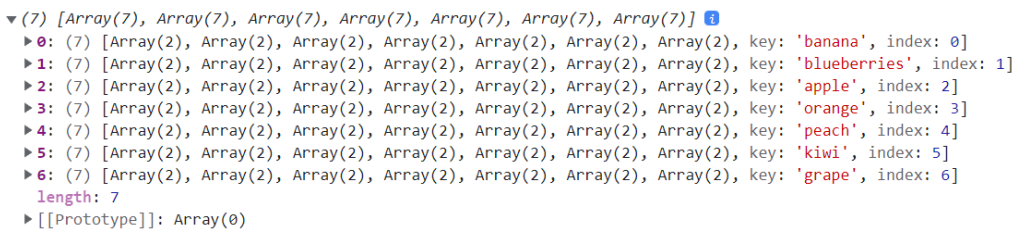
# 1. Create a stacked variable “series”

Download exercise7.html. In this lecture exercise, we will draw a stacked barchart.

**Question** *Submitted May 16th 2023 at 5:09:24 pm*

Define a stack generator `stack` , and define a variable `result` which generates a stacked object of the dataset with respect to "fruit type". The fruit types are already defined as an array `fruit`. You can refer to today’s lecture slides to get the data in a stacked layout.

If you output `result` to console, it will show




```
const stack = d3.stack()
    .keys(["banana",
           "blueberries",
           "apple",
           "orange",
           "peach",
           "kiwi",
           "grape"]);
const result = stack(data);
```

## 2. Group bars with respect to the secondary Key using <g>.

Create <g> with respect to the secondary Key (7 <g>s each of which represent a type of fruit) then fill the stacked bars with different colors for each fruit type. We already gave you the defined `colorScale`.

You can do so by grouping all the bars with the same fruit type under a single <g> then setting the color for each <g>.

Following code generates <g> for each fruit type. Notice that you do data binding with **`result`**, the nested array, not dataset.

```
const groups = svg.selectAll(".gbars")
                .data(result).enter().append("g")
                .attr("class","gbars")
```

Check if 7 <g>s have been generated on the Elements tab on the Inspect tool, along with the correct colors.

**Question** *Submitted May 16th 2023 at 5:12:50 pm*

Finish "Activity 2" in exercise7.html. Copy and paste your corresponding codes below.

```
    const groups = svg.selectAll(".gbars")
        .data(result).enter().append("g")
        .attr("class","gbars")
        .attr("fill", d => colorScale(d))
```

## 3. Draw bars for each secondary Key value (apples, oranges, and grapes)

Now you need to add bars for each <g>. All you need to do is extend the method chain `groups`. Following code adds <rect> by n times for each <g>; n is 5 in this case as there are 5 elements - which corresponds to 5 weekdays - under each <g>.

```
groups.selectAll("rect")
      .data(d => d).enter().append("rect")
```

> You are doing **nested data-binding**: `groups` was already an output after a data-binding with `stacked`, and now you are using data-binding again over `groups`. This works because `stacked` is a nested array (i.e. an array of arrays) - so there are two directions you can bind `stacked`.

The output should generate 5 <rect>s for each <g> as follows.


**Question** *Submitted May 16th 2023 at 5:31:25 pm*

Let's add x, y, width, and height attributes for each bar. Note that `d` below refers to each array of size 2.

```
groups.selectAll("rect")
      .data(d => d).enter().append("rect")
```

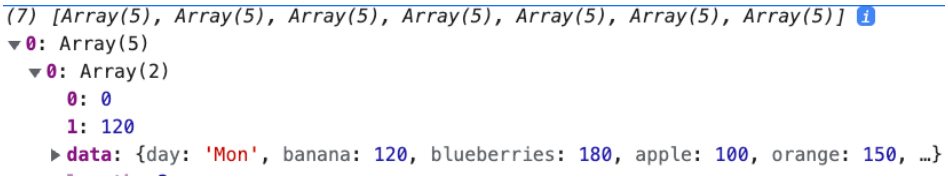
More specifically, each d refers to [0, 120], [0,60], [0, 100], etc.

```
[[0, 120],   [0, 60],    [0, 100],   [0, 80],    [0, 120],  //bananas
 [120, 300], [60, 245],  [100, 315], [80, 310],  [120, 360] //blueberries
 [300, 400], [245, 350], [315, 425], [310, 415], [360, 465] // apples
 ...
]
```

The **x** attribute should use  xScale you defined.

```
//Scale for x- and y-axis
const xScale = d3.scaleBand()
  .domain(data.map((d) => d.day)) // data.map create an array of days
  .range([0, width])
  .padding(0.1);
```

Take a look at what `groups` look like on the Inspect Tool (I put the screenshot below too) and define x using xScale.

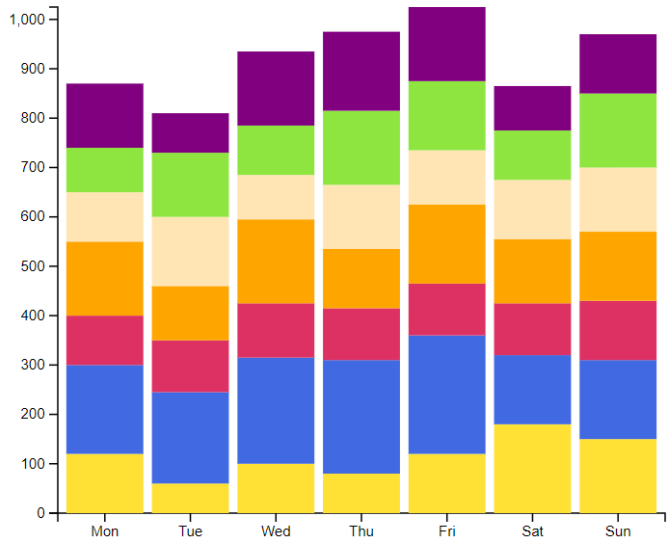



**y** depends on d[0] since each d has two values, d[0] and d[1].

**width** should use [xScale.bandwidth()]

**height** is all yours! Check how the height of each bar is calculated, e.g. oranges on Monday, grapes on Wednesday; then see if the way you calculated those bars can be generalized.

The expected output should be like this:

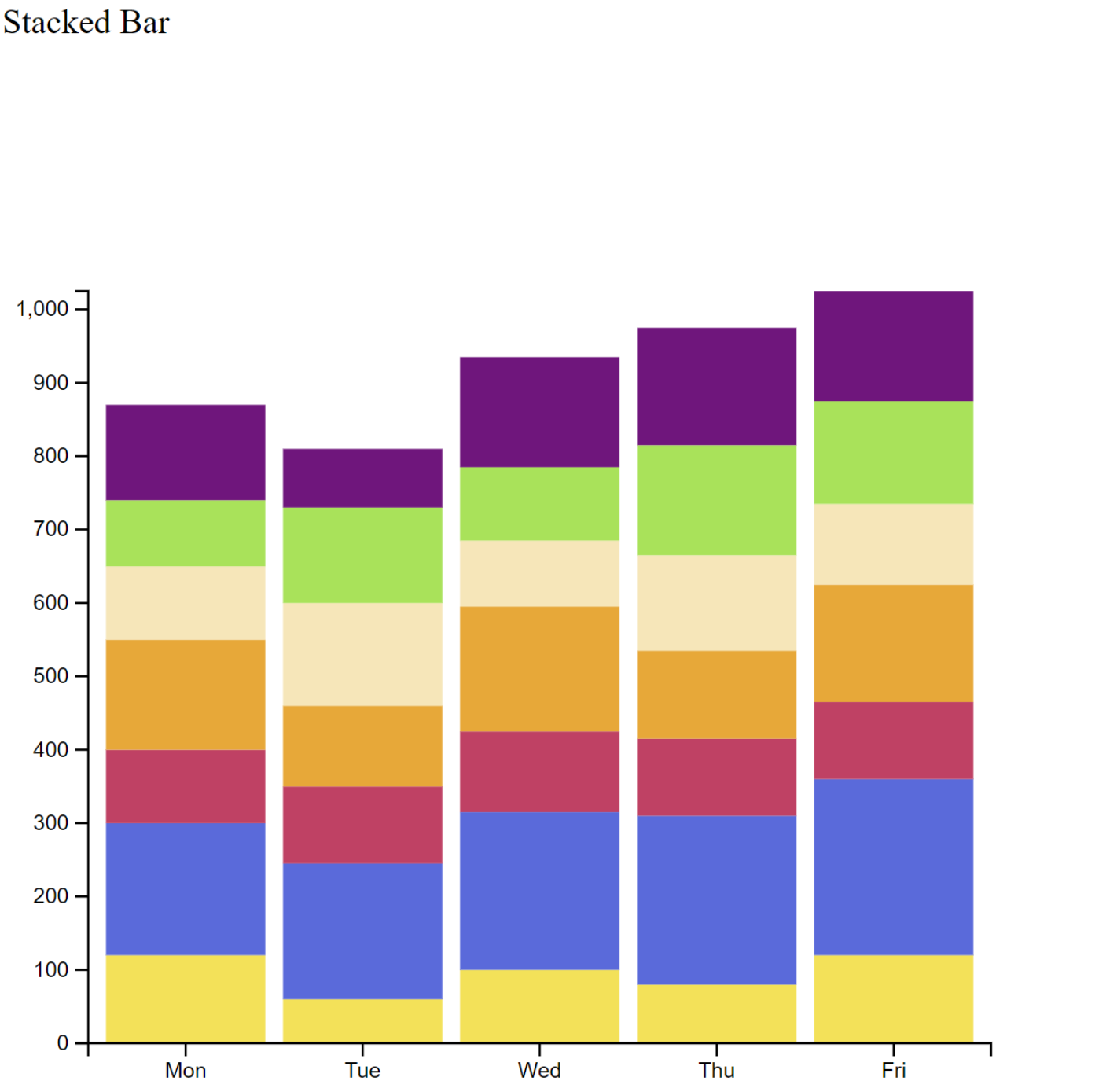



```
    groups.selectAll("rect")
      .data(d => d).enter().append("rect")
      .attr("x", d => xScale(d.data.day))
      .attr("y", d => yScale(d[1]))
      .attr("height", d => yScale(d[0]) - yScale(d[1]))
      .attr("width", xScale.bandwidth())
```

# Upload Your Files

**Question 1** *Submitted May 16th 2023 at 5:31:44 pm*

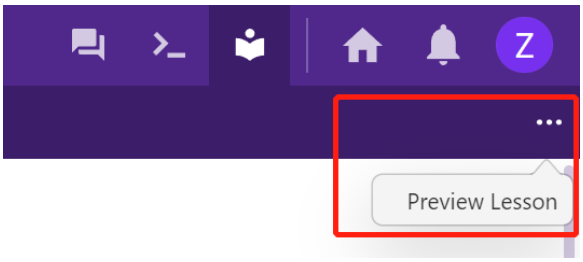
Upload the screenshot of your resulting webpage. You will need to click the "clip" button to upload a file into the Answer box.



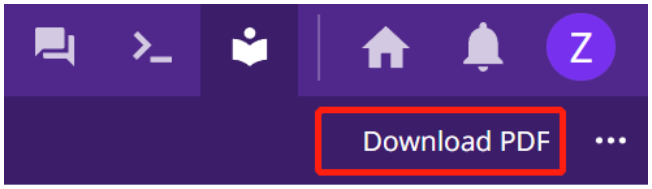
**Question 2**

You need to download the PDF of lecture exercise 6 and upload it with other files to the Gradescope. Follow the instructions on how to download PDF file:

1. Click on the ellipsis button and the Preview Lesson.




2. After that, click on the Download PDF button.




- ( ) PDF downloaded!
- ( ) Haven't done yet!

**Question 3**

Upload the following files to Gradescope. You need to make **a group submission,adding all present members in your team**, so that the present members get the participation credit.

Files to upload:

- exercise7.html
- PDF you downloaded as Q2

- ( ) Our team uploaded the the files on gradescope!
- ( ) Oops, our team did not upload the files on gradescope!

# Feedback

**Question**

Was the activity today clear? If not, please share how the course can improve it. Your comments will help us design future lab content (and also future students).

*No response*

## Example Solution

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <script src="https://d3js.org/d3.v7.min.js"></script>
    <title>Lecture Exercise 7 Material</title>
</head>
<body>

    <div id="stacked">
        <p>Stacked Bar</p>
    </div>
    <script type="text/javascript">

        // input dataset
    const data = [
            {
              day: "Mon",
              banana: 120,
              blueberries: 180,
              apple: 100,
              orange: 150,
              peach: 100,
              kiwi: 90,
              grape: 130,
            },
            {
              day: "Tue",
              banana: 60,
              blueberries: 185,
              apple: 105,
              orange: 110,
              peach: 140,
              kiwi: 130,
              grape: 80,
            },
            {
              day: "Wed",
              banana: 100,
              blueberries: 215,
              apple: 110,
              orange: 170,
              peach: 90,
              kiwi: 100,
              grape: 150,
            },
            {
```

```
          day: "Thu",
          banana: 80,
          blueberries: 230,
          apple: 105,
          orange: 120,
          peach: 130,
          kiwi: 150,
          grape: 160,
        },
        {
          day: "Fri",
          banana: 120,
          blueberries: 240,
          apple: 105,
          orange: 160,
          peach: 110,
          kiwi: 140,
          grape: 150,
        },
        {
          day: "Sat",
          banana: 180,
          blueberries: 140,
          apple: 105,
          orange: 130,
          peach: 120,
          kiwi: 100,
          grape: 90,
        },
        {
          day: "Sun",
          banana: 150,
          blueberries: 160,
          apple: 120,
          orange: 140,
          peach: 130,
          kiwi: 150,
          grape: 120,
        },
      ];

    // setting the color for each secondary Key value (types of fruits - apples, oranges, grape
const colors = [
  "#FFE135",
  "royalblue",
  "#de3163",
  "orange",
  "#FFE5B4",
  "#8ee53f",
  "purple",
];
const colorScale = d3.scaleOrdinal().domain([0, 1, 2, 3, 4, 5, 6]).range(colors);

const margin = { top: 100, bottom: 50, left: 40, right: 40 };
```

```
    const width = 500 - margin.left - margin.right;
    const height = 500 - margin.top - margin.bottom;
    const fruit = Object.keys(data[0]).filter((d) => d != "day"); // fruit array, use it to create

        /* Activity 1: stack secondary Key values, using stack() method */
    const series = d3.stack().keys(fruit);
    const stacked = series(data);

    //svg canvas
    const svg = d3
      .select("#stacked")
      .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})`);

    const xScale = d3.scaleBand()
      .domain(data.map((d) => d.day)) // data.map create an array of days
      .range([0, width])
      .padding(0.1);

    // Calculate the maximum y value for the y scale
    let maxValue = d3.max(stacked, (d) => d3.max(d, (d) => d[1]));
    const yScale = d3.scaleLinear().domain([0, maxValue]).range([height, 0]);

    // Axes
    const xAxis = d3.axisBottom().scale(xScale);
    const yAxis = d3.axisLeft().scale(yScale);
    svg.append("g").attr("transform", `translate(0, ${height})`).call(xAxis);
    svg.append("g").call(yAxis);

        /* Activity 2: group <g> bars with respect to the secondary Key i.e Fruit, and fill it with
    const groups = svg.selectAll(".gbars")
        .data(stacked).enter().append('g')
        .attr('class','gbars')
        .attr('fill', (d, i) => colorScale(i));

        /* Activity 3: draw a bar for each Key value */
    groups.selectAll("rect").data(d => d).enter().append("rect")
      .attr("x", (d) => xScale(d.data.day))
      .attr("y", (d) => yScale(d[1]))
      .attr("height", (d) => yScale(d[0]) - yScale(d[1]))
      .attr("width", xScale.bandwidth());
    </script>

</body>
</html>
```