CDS 6324
Lab 12: Dashboard with Multiple Coordinated View

Learning Objectives

- Learn to create a multiple coordinated view, than can form a nice dashboard.

Create a Dashboard with Multiple Coordinated Views

One interactive visualization is nice, multiple coordinated ones are better. By creating a dashboard with multiple coordinated views, and combined it with filtering and linking and brushing, it enables users to explore datasets and discover new insights.

This lab is adapted from **A D3 v7 Tutorial**: https://github.com/sgratzl/d3tutorial

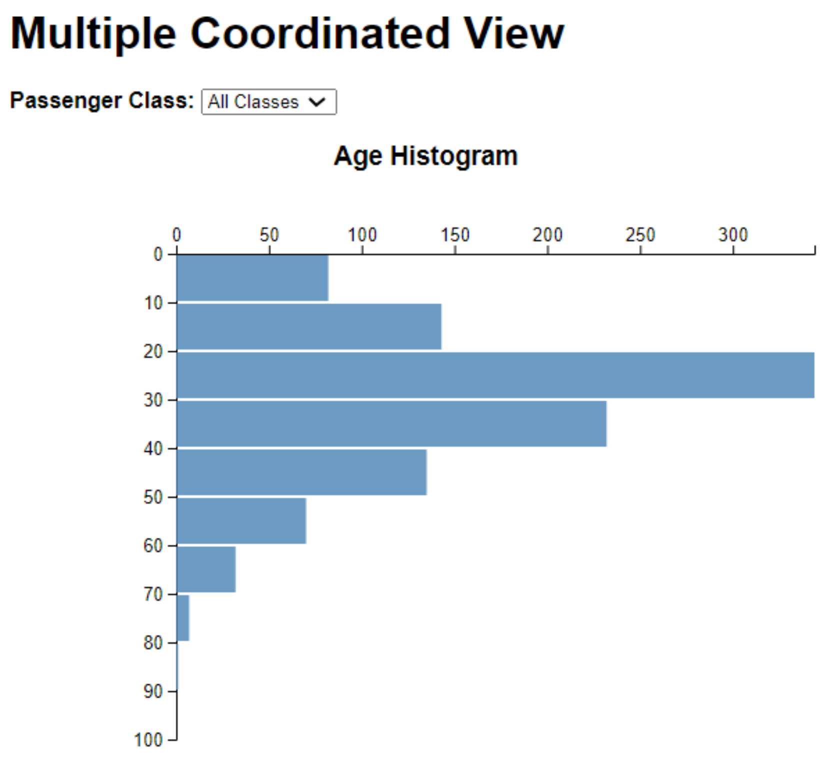
Code Structure

Before creating a multiple coordinated view, setting up a proper code structure helps. A possible way to structure ones code based on its function is as follows:

```
const state = {
  data: [],
  // e.g., user selection
}

function filterData() {
  // filter the raw data according to user selection
}

function wrangleData(filtered) {
  // wrangles the given filtered data to the format required by the
visualizations
}

function createVis() {
  // initialized for creating the visualizations, e.g., setup SVG, init
scales, ...

function update(new_data) {
  // updates the specific visualization with the given data
}

  // return the update function to be called
  return update;
}

// create a specific instance
```

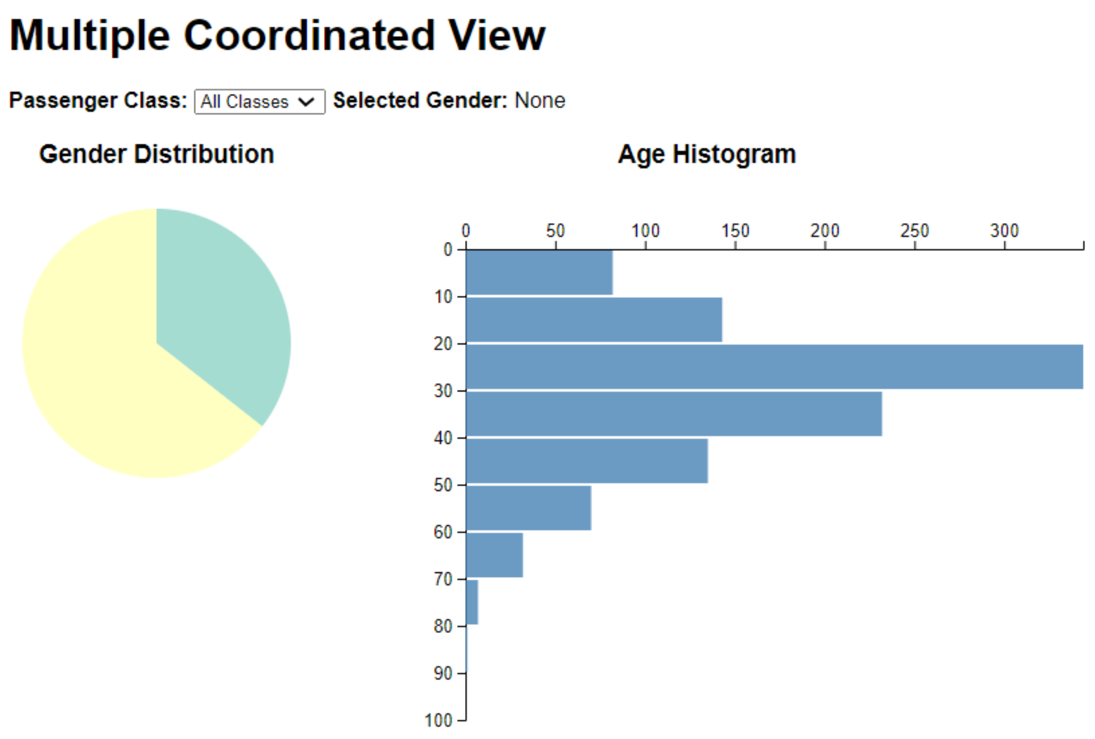
```
const vis = createVis();

function updateApp() {
  // updates the application

  const filtered = filterData();
  const new_data = wrangleData(filtered);

  // update visualization
  vis(new_data);
}

// init interaction, e.g., listen to click events
d3.select(...).on('click', () => {
  // update state
  updateApp();
})

d3.json(...).then((data) => {
  // load data, e.g., via d3.json and update app afterwards
  state.data = data;
  updateApp();
});
```

Now, open the file **mcv01_initial.html**. This program uses data from **titanic3.csv** to draw a bar chart showing the age of the victims of Titanic with a passenger class filter based on the above code structure as shown below. Go through the program and try to understand how it works.



Interaction and Filtering

Next, we create a pie chart and allow users to intuitively interact with the visualization directly. At the end of this exercise, we will obtain the following output, where filtering can be performed via the ‘Passenger Class’ drop-down menu or by clicking the the pie chart to filter by sex:



The step-by-step instructions to add in a pie chart to create a dashboard with two coordinated views are given below.

Step 1: Add the gender distribution filter:

```
<section>
  <h3>Gender Distribution</h3>
  <svg id="sex"></svg>
</section>
```

Step 2: Add a function to create a pie chart

```
function createPieChart(svgSelector) {
  const margin = 10;
  const radius = 100;

  // Creates sources <svg> element
  const svg = d3
    .select(svgSelector)
    .attr("width", radius * 2 + margin * 2)
    .attr("height", radius * 2 + margin * 2);
```

```
        // Group used to enforce margin
        const  g  =  svg.append("g").attr("transform",  `translate(${radius  +
margin},${radius + margin})`);

        // 1. TODO

        function update(new_data) {
          //{key: string, values: IPerson[]}[]
          // TODO apply layout and render pie
        }

        return update;
      }
```

Step 3: Add a pie chart with simple interaction that allow the selection on the ‘Passenger Class’ to change both the pie chart.

```
      function createPieChart(svgSelector) {
        const margin = 10;
        const radius = 100;

        // Creates sources <svg> element
        const svg = d3
          .select(svgSelector)
          .attr("width", radius * 2 + margin * 2)
          .attr("height", radius * 2 + margin * 2);

        // Group used to enforce margin
        const  g  =  svg.append("g").attr("transform",  `translate(${radius  +
margin},${radius + margin})`);

        const pie = d3
          .pie()
          .value((d) => d.values.length)
          .sortValues(null)
          .sort(null);
        const arc = d3.arc().outerRadius(radius).innerRadius(0);
        const cscale = d3.scaleOrdinal(d3.schemeSet3);

        function update(new_data) {

          const pied = pie(new_data);

          // Render the chart with new data
          cscale.domain(new_data.map((d) => d.key));

          // DATA JOIN
          const path = g
            .selectAll("path")
            .data(pied, (d) => d.data.key)
            .join(
              // ENTER
              // new elements
              (enter) => {
                const path_enter = enter.append("path");
                // TODO register click handler to change selected sex in
                // state and call updateApp()
                path_enter.append("title");
                return path_enter;
              }
            );
```

Step 4: Add the following code to the function wrangleData (before return) and return both sets of data (replace the return code).

```
const sexPieData = ["female", "male"].map((key) => ({
          key,
          values: filtered.filter((d) => d.sex === key),
}));

// replace the current return line of code
return { ageHistogramData, sexPieData };
```

Step 5: Create a variable to call the function createPieChart.

```
const sexPieChart = createPieChart("#sex");
```

Step 6: Replace/Add the following code to the updateApp() function:

```
// Replace the following two lines of code in the original program
const { ageHistogramData, sexPieData } = wrangleData(filtered);
ageHistogram(ageHistogramData)

// Add the following code to call the createPieChart function via the constant
// variable sexPieChart
sexPieChart(sexPieData);
```

Step 7: Ass the direct interaction to allow users to filter the charts in the dashboard by clicking on a sex category on the pie chart. To return to the original complete pie chart, users can just click on the pie chart again.

Add the following code to the <div> </div> tag to show the selected sex:

```
<strong>Selected Gender: </strong>
<span id="selectedSex"></span>
```

Replace the initial value of the **state** variable to set the initial selectedSex to null

```
const state = {
        data: [],
        passengerClass: "",
        selectedSex: null
      };
```

Add the following code to the **path** constant variable **createPieChart**() function before the "**return path_enter;**" statement. This is the key code that changes the dataset to utilize only the data associated with the selected sex.

```
                path_enter.on("click", (e, d) => {
                  if (state.selectedSex === d.data.key) {
                    state.selectedSex = null;
                  } else {
                    state.selectedSex = d.data.key;
                  }
                  updateApp();
                });
```

Replace the path settings code as follow:

```
          path
            .classed("selected", (d) => d.data.key === state.selectedSex)
            .attr("d", arc)
            .style("fill", (d) => cscale(d.data.key));
```

Add the following code to the **filterData()** function before the return statement:

```
          if (state.selectedSex && d.sex !== state.selectedSex) {
            return false;
          }
```

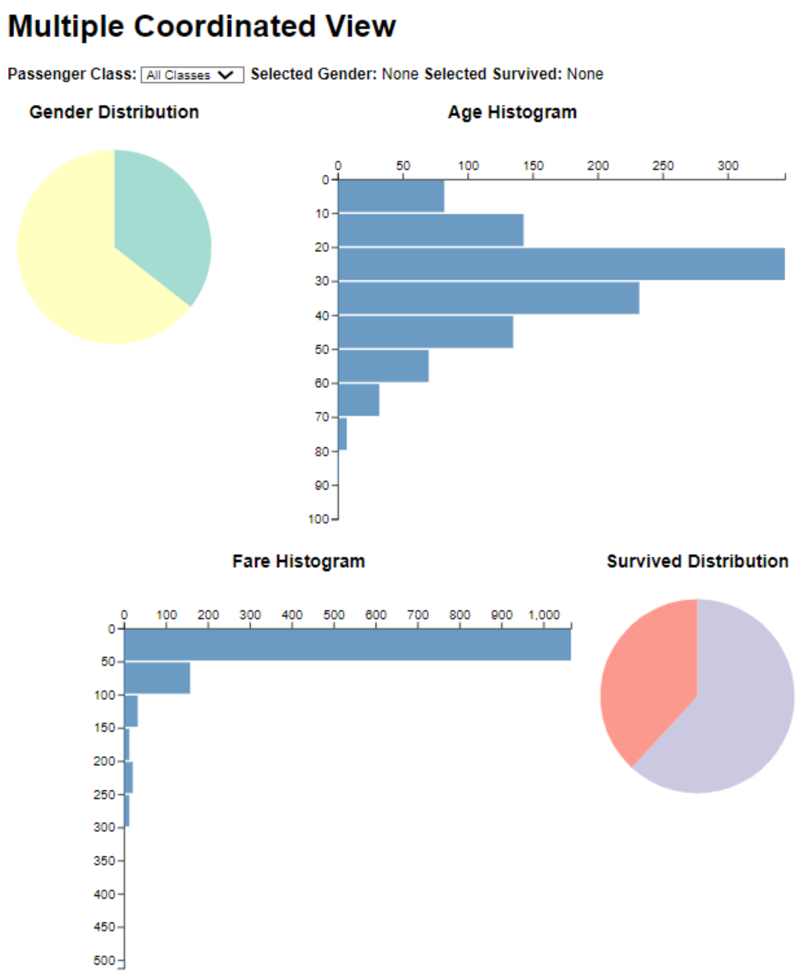
Add the following code as the last line of code to the **updateApp()** function:

```
          d3.select("#selectedSex").text(state.selectedSex || "None");
```

Reusability

The advantage of the above code structure is that we can use the factory methods to create multiple instances of the same visualization kind showing different aspect of the data. This is a simple yet effective way to improve the overall multiple coordinated setup.

Exercise: Extending the above program with two coordinated views, add another two charts to create a dashboard with four corrdinated charts as shown below:


Multiple Coordinated View
Passenger Class: All Classes Selected Gender: None Selected Survived: None
Gender Distribution
Age Histogram
Fare Histogram
Survived Distribution

SUBMISSION

Submit the completed Exercise to eBWise with the filename <StudentID>_Lab12.html or folder name <StudentID>_Lab12 if you have more than one file. The marks for this exercise will contribute towards "Class Participation" assessment. The submission deadline is on **23rd June, 2023, 11:59pm**
.

More Examples from the Web

a) **Coordinated Views with Brushing and Linking**
https://observablehq.com/@yurivish/coordinated-views

b) **Using D3 Dispatch to Link Multiple Views**
https://gist.github.com/davegotz/c15f68fb7a50c867801373bd2f864fdc