Next D3.js Assignment: Creating a Connected Multi-View Dashboard

Assignment Overview

Objective: Build an interactive data dashboard that combines three coordinated visualization components:

1. An interactive datatable with filtering and sorting
2. A time series area chart with brushing functionality
3. A stacked bar chart (from previous assignment)

Dataset: Chocolate Sales Data from Kaggle (same as previous assignment)

Time: Approximately 45-60 minutes

Difficulty: Intermediate

Assignment Structure

This assignment builds on the previous exercise by adding new visualization components and implementing cross-component interactions (brushing and linking). Students will implement a dashboard where selecting data in one visualization will highlight or filter related data in the others.

Step-by-Step Implementation with Structured Prompts

Step 1: Create an Interactive Datatable

Use this prompt as a base to generate the datatable component:

```
"Create a D3.js datatable for the Chocolate Sales data that:

1. Data Processing:
   - Loads the cleaned CSV data we prepared previously
   - Implements column-based sorting (ascending/descending) when clicking headers
   - Enables text-based filtering for each column

2. Visual Design:
   - Creates a clean, responsive table layout with appropriate styling
```

- Highlights rows on hover for better readability
- Displays pagination if the dataset exceeds a certain size (e.g., 10 rows)

3. Interaction:
- Implements click events on rows to select/highlight specific data points
- Stores selected row information for cross-visualization communication
- Provides visual feedback for selected rows

Please include detailed comments explaining key implementation decisions."

Step 2: Implement Time Series Area Chart with Brushing

Use this prompt to generate the area chart with brushing functionality:

"Implement a D3.js time series area chart for the Chocolate Sales data that:

1. Data Processing:
- Aggregates sales data by month across all products/categories
- Formats dates properly on the x-axis
- Calculates cumulative sales values for the area chart

2. Visual Design:
- Creates a main area chart showing sales trends over time
- Adds a smaller 'context' area chart below for navigation/brushing
- Uses an appropriate color scheme that matches our existing visualizations

3. Brushing Functionality:
- Implements d3.brushX() to allow users to select a time range
- Updates the main chart to zoom into the selected time period
- Maintains the context chart to show the complete timeline
- Provides smooth transitions when the brush selection changes

Please structure your code with clear separation between data processing, visualization creation, and interaction handling."

Step 3: Connect the Visualization Components

Use this prompt to connect all the visualization components:

"Develop the code needed to connect our three visualization components (datatable, area chart, and stacked bar chart) into a coordinated dashboard:

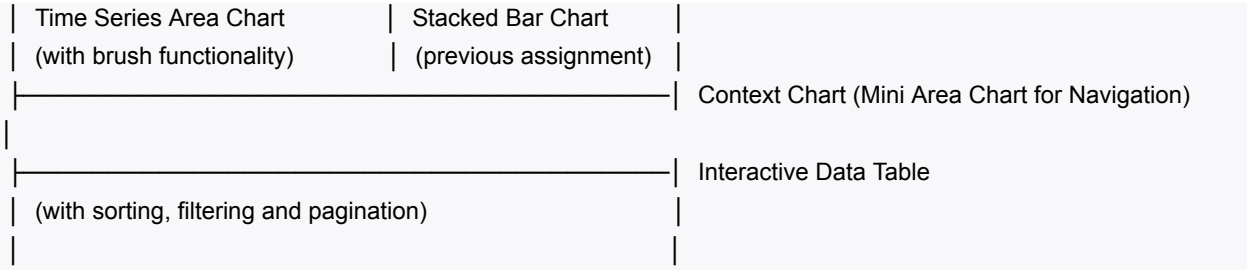
1. Cross-Filtering Implementation:
   - When a time range is selected in the area chart, filter both the datatable and bar chart to that period
   - When a row is selected in the datatable, highlight corresponding data in both charts
   - When a category is selected in the bar chart, filter the table and highlight the area chart accordingly

2. State Management:
   - Implement a simple state management system to track selections across components
   - Create update functions for each visualization that respond to state changes
   - Ensure consistent visual feedback across all components when selections change

3. User Experience:
   - Add a 'Reset' button to clear all filters and selections
   - Implement smooth transitions when views update
   - Ensure the dashboard layout is responsive and components resize appropriately

Please include detailed comments explaining the communication mechanism between visualization components."

Dashboard Layout Recommendation



Evaluation Criteria

1. **Component Implementation**: Correct implementation of datatable and area chart with brushing
2. **Visual Design**: Cohesive visual design across all dashboard components
3. **Interaction Implementation**: Effective brushing and linking between visualizations
4. **Code Organization**: Well-structured code with clear separation of concerns
5. **Performance**: Efficient data handling and smooth transitions

6. **Cross-Component Communication**: Clean implementation of the coordination between visualizations

Technical Concepts to Master

For this assignment, students should understand:

1. **Brushing and Linking**: The technique of selecting elements in one view to highlight related data in other views
2. **D3.js Brush Component**: Using `d3.brushX()` for time-based selection
3. **Focus+Context Visualization**: Implementing the overview+detail pattern with the mini context chart
4. **Event Handling**: Propagating selection events between visualization components
5. **State Management**: Tracking the current selection state across multiple visualizations

Sample Code for Cross-Component Communication

```
// Simple state management object
 const appState = {
  timeRange: null,            // Selected time range from brush
  selectedProduct: null, // Selected product category
  selectedRows: [],           // Selected rows from datatable

  // Update state and notify all visualizations
  updateTimeRange(range) {
          this.timeRange = range;
          this.notifyVisualizations();
  },

  updateSelectedProduct(product) {
          this.selectedProduct = product;
          this.notifyVisualizations();
  },

  updateSelectedRows(rows) {
    this.selectedRows = rows;
          this.notifyVisualizations();
  },

  notifyVisualizations() {
```

```
            // Update each visualization based on current state
            updateAreaChart(this);
            updateBarChart(this);
            updateDataTable(this);
  }
};

// Example of brush handler that updates state
function handleBrush(event) {
  if (!event.selection) return;
  const [x0, x1] = event.selection.map(x => xScale.invert(x));
  appState.updateTimeRange([x0, x1]);
}
```