Interactive Multi-Dimensional Housing Data Visualization with Coordinated Views

Jan Esquivel Marxen
University of Luxembourg
Luxembourg
jan.esquivel.001@student.uni.lu

Abstract

Effective visualization of multi-dimensional datasets requires coordinated interaction patterns that enable users to explore relationships between continuous and categorical variables. This report presents an interactive visualization system for housing market data implementing synchronized scatterplot and violin plot views with multi-rectangle brushing. The left view displays area-price relationships through a traditional scatterplot, while the right view presents price distributions across categorical counts (bedrooms, bathrooms, stories) using kernel density estimation. The system demonstrates React-D3.js integration patterns with parameterized violin rendering supporting n variables, dynamic color palettes, and persistent multi-rectangle selection enabling non-linear subset queries. We justify the visual encodings based on data properties and user tasks, discuss architectural decisions including the React container pattern and shared state management, and evaluate the trade-offs of design choices including KDE bandwidth selection and multi-brush interaction complexity.

Note: Portions of this report were developed with the assistance of artificial intelligence tools to support code generation, documentation, and readability improvements. All technical content and implementations were reviewed and validated by the author.

Keywords: Data Visualization, D3.js, React, Information Visualization, Kernel Density Estimation, Violin Plots, Coordinated Views, Brushing and Linking

ACM Reference Format:
Jan Esquivel Marxen. 2025. Interactive Multi-Dimensional Housing Data Visualization with Coordinated Views. In *Proceedings of Data Visualization Course (Data Viz 25).* ACM, New York, NY, USA, 3 pages.

1 Introduction

Exploratory analysis of multi-dimensional datasets benefits from visualization techniques that reveal patterns across both continuous and categorical variables while supporting interactive investigation. Housing market data presents a rich domain for visualization: prices depend on continuous variables (property area) and discrete structural characteristics (bedrooms, bathrooms, stories). Traditional scatterplots show bivariate relationships effectively but provide limited insight into distributional patterns across categories.

This project implements coordinated multiple views combining a scatterplot with violin plot distributions:

Left View - Scatterplot with 2D Brush:

- Area vs. Price with multi-rectangle persistent brushing
- Manual rectangle creation supporting complex queries
- Click selection for individual data point inspection

Right View - Violin Plots with Interaction:

- Price distributions for bedrooms, stories, bathrooms (1-5)
- Kernel density estimation with Epanechnikov kernel
- Jittered points overlay within violin contours
- Multi-rectangle brushing synchronized with left scatterplot
- Parameterized design supporting n categorical variables

Both views implement synchronized highlighting: selections in either view highlight respective points in both. The architecture follows React-D3 best practices with separate container components managing state and D3 classes handling rendering.

This report justifies visual encodings based on data properties and exploratory tasks (Section 2), describes the React-D3 architecture and implementation patterns (Section 3), and discusses design trade-offs including KDE parameters, color encoding, and multi-brush complexity (Section 4).

2 Visual Design and Justification

2.1 Dataset Properties

The Housing.csv dataset contains 545 records describing residential properties with the following attributes:

- **Continuous:** price (dependent variable), area (square feet)

- **Discrete ordinal:** bedrooms, bathrooms, stories (counts 1-5)
- **Categorical:** furnishing status (furnished, semi-furnished, unfurnished)

User Tasks: The visualization supports exploratory analysis tasks: (1) identifying area-price correlation patterns, (2) comparing price distributions across bedroom/bathroom/story counts, (3) investigating outliers and their categorical characteristics, (4) selecting subsets matching non-linear criteria (e.g., "3-bedroom houses priced 2M-4M").

2.2 Scatterplot Design (Left View)

Encoding Rationale:

- **X-axis (area):** Linear scale for continuous quantitative variable; position is the most effective channel for quantitative data
- **Y-axis (price):** Linear scale shared between left and right views to enable direct comparison

Interaction Design:

- **Multi-rectangle brushing:** Users draw persistent rectangles via mouse movements and clicks. Multiple rectangles accumulate, enabling union queries (e.g., select both low-area/low-price AND high-area/high-price segments)
- **Click selection:** Single-click on points for individual inspection
- **Clear selection button:** Removes all brushes and resets selection state

: expand pros and cons
Design Pros: Familiar scatterplot idiom; multi-rectangle brushing supports more complex queries than only single-rectangle selection; position encoding maximizes precision.

Design Cons: Overplotting possible at high densities; no categorical variable encoding (e.g., furnishing status not shown); multi-brush UI requires learning.

2.3 Violin Plot Design (Right View)

Encoding Rationale:

- **X-axis:** Categorical band scale for bedroom/bathroom/story counts (1-5). Each x-position represents a count value; n violins are positioned evenly within each band.
- **Y-axis:** Price (shared scale with left view, no redundant axis drawn)
- **Violin contour:** Area encodes kernel density estimates of price distributions per category. Width at a given y-value represents relative frequency of that price.
- **Points (jittered):** Individual data points overlaid within violin width to provide access to raw data.
- **Color:** D3 Tableau10 palette assigns distinct hues to bedrooms/bathrooms/stories.
- **Separators:** Dashed vertical lines between x-categories improve visual grouping.

Interaction Design: Same pattern as left view; rectangles select points within violin plots. Selections update both views simultaneously.

Pros: Violin plots show full distributions (not just summary statistics like box plots); KDE reveals multi-modal patterns; color-coded variables enable side-by-side comparison; points provide individual data access; parameterized design supports adding/removing variables dynamically.

Cons: Violin plots less familiar than box plots; KDE bandwidth parameter affects smoothness (requires tuning); horizontal scrolling needed for many variables; multi-modal distributions may be harder to interpret than histograms for some users.

2.4 Design Alternatives Considered

Parallel Coordinates: Would show all variables simultaneously but sacrifice distributional detail and suffer from overplotting at 545 records. Would also require scrolling for many variables. Is already well-known, whereas our goal was to show alternative visualizations.

Heatmap/Matrix: Could show bivariate distributions but would not preserve individual data points and requires binning choices (not the point of this assignment).

Small Multiples of Histograms: Similar to violins but less compact; histograms require bin width tuning and lose smooth distributional curves. We would also not be able to overlay individual points easily.

The chosen design balances distributional insight (violins), individual data access (points), and effective multivariate exploration (scatterplot) while supporting non-linear interactive queries (multi-brush).

3 Implementation and Architecture

3.1 React-D3 Integration Pattern

The system follows a clean separation between React (declarative state management) and D3 (imperative SVG rendering):

```
App.js (React root)
  +-- ScatterplotContainer (React)
  |     +-- Scatterplot-d3.js (D3 class)
  +-- ViolinScatterContainer (React)
        +-- ViolinScatter-d3.js (D3 class)
```

App.js responsibilities:

- CSV loading via `fetchCSV` utility
- Compute shared `yDomain` for synchronized price scales
- Manage `selectedItems` state (array of selected data objects)
- Provide `scatterplotControllerMethods` callbacks (`updateSelectedItems`, `handleOnBrush`)
- Pass `violinVariables` array to right view

Container components:

- Create D3 class instance on mount (`useEffect`)
- Forward data, controller methods, and configuration to D3
- Compute color palettes from variable arrays (`useMemo`)
- Render legends and UI controls (Clear button)
- Call `highlightSelected` when `selectedItems` changes

D3 classes:

- `create(config)`: Build SVG structure, groups, pointer event handlers
- `render(data, controllerMethods)`: Compute scales, KDE, draw marks
- `highlightSelected(selectedItems)`: Update opacity/stroke for selected items
- `clear()`: Cleanup on unmount

This pattern avoids common pitfalls: D3 instances are created once (not on every render), callbacks prevent infinite re-render loops, and imperative D3 logic stays isolated from React's declarative model.

3.2 Multi-Rectangle Brushing

Unlike `d3.brush` (single brush area), we implement persistent multi-rectangle creation:

```
svgElem.on('mousedown.multiRect', (event) => {
  const [mx,my] = d3.pointer(event, this.svg.node());
   this._creating = true;
   this._createStart = [mx,my];
   this._currentRect = this.svg.append('rect')
     .attr('class','multi-brush')
     .attr('x', mx).attr('y', my)
     .attr('width', 0).attr('height', 0);
});
```

On `mouseup`, the rectangle's bounding box is stored in `this.multiBrushes`, and `updateBrushesSelection()` computes the union of points inside all rectangles. Selected items are passed to React via `controllerMethods.handleOnBrush(unique)`, triggering `highlightSelected()` in both views.

4 Discussion and Trade-offs

4.1 Multi-Brush Interaction Complexity

Pros: Enables more complex queries (union of multiple ranges); persistent rectangles provide visual feedback of selection criteria.

Cons: Non-standard interaction pattern; users must learn mousedown-drag-mouseup workflow; no per-rectangle delete UI (only global Clear); overlapping rectangles could confuse users.

Improvement: Adding individual 'x' buttons to each rectangle or implementing shift-click to toggle rectangles would improve usability; being able to zoom/pan the scatterplot would also help.

4.2 Color Encoding vs. Small Multiples

Choice: Three variables (bedrooms/bathrooms/stories) are color-coded and overlaid within each x-category.

Pros: Compact layout; direct side-by-side comparison; color legend links variables to hues.

Cons: Overlapping violins may occlude each other; color discrimination harder than spatial separation; limited to 10 variables (palette size), unless we repeat colors.

Alternative: Small multiples (separate panels per variable) would eliminate occlusion but sacrifice compactness and would make cross-variable comparison harder.

5 Conclusion

This project demonstrates effective patterns for interactive multi-dimensional visualization with React and D3.js. The coordinated scatterplot-violin view combination enables rich exploratory analysis of housing data, revealing both bivariate relationships and distributional patterns across categories. Multi-rectangle brushing supports complex subset queries, and the parameterized architecture allows dynamic variable selection.

Key contributions:

- Clean React-D3 integration avoiding re-render pitfalls
- Parameterized violin plot architecture supporting n variables
- Multi-rectangle brushing for flexible subset selection
- KDE utilities (helper.js) for reusable density estimation
- Comprehensive inline documentation for maintainability

The design balances distributional insight (KDE violins), individual data access (jittered points), and effective bivariate exploration (scatterplot) while following established React patterns (useState, useRef, useEffect, useMemo) and D3 idioms (scales, generators, join). Future work could improve KDE bandwidth selection, add per-rectangle deletion, and optimize performance for larger datasets.