FWA Model Dashboard Recommendations

Based on your requirements, here are comprehensive dashboard suggestions for your Fraud, Waste, and Abuse (FWA) monitoring system:

1. Executive Summary Dashboard

* **KPI Cards**: Total claims, flagged claims, potential savings
* **Risk Distribution**: Pie chart showing clean vs. outlier claims
* **Savings Opportunity**: Gauge showing potential recoverable amount
* **Top Alerts**: Highest priority outliers needing review

2. KRI Outlier Analysis Dashboard

* **Outlier Flag Summary**: Table/grid showing all KRIs with outlier counts
* **Severity Heatmap**: Color-coded by risk level (high/medium/low)
* **Outlier Trend**: Line chart showing outlier frequency over time
* **Drill-down**: Ability to click into specific KRI details

3. Assessor vs Model Decision Dashboard

* **Comparison Matrix**: Grid showing agreement/disagreement by KRI
* **Claim Volume**: Bar chart comparing assessor vs model decisions
* **Financial Impact**: Stacked bar showing billed vs reject amounts
* **Discrepancy Analysis**: Reasons for decision mismatches

4. Provider Watchlist Dashboard

* **High-Risk Providers**: Sortable table with risk scores
* **Provider Network Map**: Geographic visualization of providers
* **Behavior Patterns**: Usage trends for watchlist providers
* **Peer Comparison**: How providers compare to specialty/region norms

5. Member Lens Dashboard (Low Priority)

* **Member Risk Scoring**: Distribution of member risk levels
* **Usage Patterns**: Frequent services, providers, conditions
* **Anomaly Detection**: Unusual billing patterns per member
* **Longitudinal View**: Member claim history timeline

6. Length of Stay Outlier Dashboard

* **LOS Distribution**: Histogram showing min/max/average
* **Facility Comparison**: LOS by hospital/provider
* **Diagnosis Analysis**: LOS outliers by diagnosis/procedure
* **Benchmarking**: Against industry/regional standards

7. Potential Savings Dashboard

* **Savings Estimation**: By KRI, provider, service type
* **Threshold Analysis**: What-if scenarios for different thresholds
* **Recovery Projections**: Monthly/quarterly potential
* **Top Opportunities**: Highest dollar value outliers

8. Clinical Attributes View

* **Diagnosis Patterns**: Frequent/most expensive conditions
* **Procedure Analysis**: Unusual combinations or frequencies
* **Medication Flags**: Off-label use, excessive quantities
* **Clinical Documentation**: Missing/insufficient documentation flags

9. Trend Analysis Dashboard

* **Temporal Patterns**: Seasonality, emerging trends
* **Comparative Trends**: Year-over-year, quarter-over-quarter
* **Risk Evolution**: How outlier patterns are changing
* **Forecasting**: Predictive modeling of future risks

10. Simulation View (Advanced)

* **Threshold Sliders**: Adjust P95, P99, custom values
* **Scenario Testing**: "What-if" analysis for policy changes
* **Impact Projections**: Financial effects of threshold changes
* **Sensitivity Analysis**: Which variables have most impact

Additional Recommendations:

1. **Cross-Filtering**: All dashboards should allow filtering by time period, region, provider type, etc.
2. **Alert System**: Integration with email/notification system for critical outliers
3. **Workflow Integration**: Ability to assign cases directly from dashboards
4. **Export Capabilities**: All views should allow data export for further analysis
5. **User Access Control**: Different views for analysts vs managers vs executives

Would you like me to elaborate on any particular dashboard or suggest specific visualization types for any of these components?

Wanted to use d3 js. Can you please provide better visuals for each section

Enhanced FWA Dashboards with D3.js Visualizations

Here are advanced D3.js visualization recommendations for each of your FWA dashboard sections:

1. Executive Summary Dashboard

**Visualizations:**

* **Interactive Sunburst Chart**: Hierarchical view of claims by risk level → category → subcategory
* **Animated Bubble Chart**: Risk severity (size) vs frequency (x-axis) vs potential savings (y-axis)
* **Dynamic Gauges**: Real-time risk meters with color thresholds (green/yellow/red)
* **Time-based Heatmap**: Daily/weekly risk patterns with brush zooming

2. KRI Outlier Analysis Dashboard

**Visualizations:**

* **Parallel Coordinates Plot**: Show correlations between multiple KRIs
* **Force-Directed Graph**: Show relationships between outlier types and providers
* **Violin Plots**: Distribution analysis for each KRI metric
* **Interactive Radar Chart**: Provider/claim risk profiles across all KRIs

3. Assessor vs Model Decision Dashboard

**Visualizations:**

* **Sankey Diagram**: Flow of claims through decision paths (model → assessor → final)
* **Diverging Bar Chart**: Side-by-side comparison of accept/reject decisions
* **Scatterplot Matrix**: Model confidence vs assessor confidence with brushing
* **Chord Diagram**: Disagreement patterns between different assessors

4. Provider Watchlist Dashboard

**Visualizations:**

* **Interactive Node-Link Diagram**: Provider networks with clustering
* **Geospatial Map**: Provider locations with risk-level overlays (Leaflet/D3 combo)
* **Horizon Charts**: Compact trend visualization for multiple providers
* **Beeswarm Plot**: Provider performance clustering by specialty/region

5. Member Lens Dashboard

**Visualizations:**

* **Streamgraph**: Temporal patterns of member claim types
* **Matrix Bubble Chart**: Member clusters by utilization patterns
* **Dynamic Tree Table**: Drill-down member hierarchy with embedded charts
* **Arc Diagram**: Member-provider relationship networks

6. Length of Stay Outlier Dashboard

**Visualizations:**

* **Box Plot with Jitter**: LOS distribution by diagnosis with actual points
* **Calendar View**: Heatmap of LOS outliers by day/month
* **Connected Scatterplot**: LOS vs cost correlation with animation
* **Violin-Box Hybrid**: LOS distributions by provider type

7. Potential Savings Dashboard

**Visualizations:**

* **Interactive Waterfall Chart**: Savings breakdown by KRI category
* **Threshold-Slider Histogram**: Adjustable cutoff visualization
* **Treemap**: Savings opportunities by provider hierarchy
* **Cumulative Curve**: Percentage of savings at different threshold levels

8. Clinical Attributes View

**Visualizations:**

* **Dendrogram**: Hierarchical clustering of diagnosis patterns
* **Circular Packing**: Nested clinical code relationships
* **Alluvial Diagram**: Treatment pathway flows
* **3D Scatterplot**: Procedure-diagnosis-cost relationships (WebGL)

9. Trend Analysis Dashboard

**Visualizations:**

* **Animated Bubble Timeline**: Emerging risks over time
* **Small Multiples**: Comparative trends across regions/KRIs
* **Stacked Area Chart**: With stream transition animations
* **Cycle Plot**: Seasonal pattern analysis

10. Simulation View

**Visualizations:**

* **Interactive Contour Plot**: Threshold sensitivity analysis
* **Dynamic Box Plot**: Live updates when adjusting percentiles
* **Parameter Network**: Show relationships between configurable variables
* **Monte Carlo Simulation**: Visualize outcome distributions

Implementation Tips:

1. **Transitions**: Use D3's enter/update/exit patterns for smooth animations
2. **Brushing/Linking**: Connect visualizations for cross-filtering
3. **Web Workers**: Offload heavy computations for large datasets
4. **Responsive Design**: Use D3's resize observer patterns
5. **Custom Interactions**:
   * Drag thresholds directly on visualizations
   * Right-click to drill down
   * Shift-brush for multivariate selection

Example Code Structure:

*// Sample D3 implementation for the Parallel Coordinates Plot*

function createParallelCoordinates(selector, data) {

const margin = {top: 30, right: 50, bottom: 30, left: 50};

const width = 800 - margin.left - margin.right;

const height = 500 - margin.top - margin.bottom;

*// Create scales for each dimension*

const dimensions = Object.keys(data[0]).filter(d => d !== 'id');

const y = {};

dimensions.forEach(dim => {

y[dim] = d3.scaleLinear()

.domain(d3.extent(data, d => +d[dim]))

.range([height, 0]);

});

*// Build the canvas*

const svg = d3.select(selector)

.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})`);

*// Add axes*

dimensions.forEach((dim, i) => {

svg.append("g")

.attr("transform", `translate(${x(dim)})`)

.call(d3.axisLeft(y[dim]));

});

*// Draw lines*

svg.selectAll(".line")

.data(data)

.enter().append("path")

.attr("class", "line")

.attr("d", path)

.style("stroke", d => color(d[dimensions[0]]));

*// Add brushing*

svg.selectAll(".dimension")

.on("brush", brushed);

}

Would you like me to provide more specific D3 code examples for any particular visualization type? I can also suggest optimization techniques for handling large healthcare datasets.

Please go ahead

D3.js Implementation Deep Dive for FWA Dashboards

I'll provide detailed implementations for key visualizations from your requirements, focusing on healthcare FWA patterns and performance optimization.

1. Parallel Coordinates Plot for KRI Outlier Analysis

**Best for:** Visualizing correlations between multiple KRIs simultaneously

javascript

function createParallelCoordinates(selector, claimsData) {

*// Data preparation - normalize KRI scores*

const dimensions = ['LOS\_score', 'cost\_anomaly', 'frequency\_flag', 'clinical\_risk'];

const normalizedData = claimsData.map(claim => {

const obj = {id: claim.claim\_id};

dimensions.forEach(dim => {

obj[dim] = (claim[dim] - minValues[dim]) / (maxValues[dim] - minValues[dim]);

});

return obj;

});

*// Set up the canvas*

const margin = {top: 30, right: 50, bottom: 30, left: 50};

const width = 1200 - margin.left - margin.right;

const height = 500 - margin.top - margin.bottom;

const svg = d3.select(selector)

.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})`);

*// Create scales for each dimension*

const x = d3.scalePoint()

.range([0, width])

.padding(0.5)

.domain(dimensions);

const y = {};

dimensions.forEach(dim => {

y[dim] = d3.scaleLinear()

.range([height, 0])

.domain([0, 1]);

});

*// Draw the lines*

const line = d3.line()

.defined(d => !isNaN(d.value))

.x(d => x(d.name))

.y(d => y[d.name](d.value));

const paths = svg.append("g")

.selectAll("path")

.data(normalizedData)

.enter().append("path")

.attr("class", "line")

.attr("d", d => line(dimensions.map(name => ({name, value: d[name]}))))

.style("stroke", d => d.fraud\_flag ? "#ff4d4d" : "#4d79ff")

.style("opacity", 0.5)

.on("mouseover", function(event, d) {

d3.select(this).style("opacity", 1).style("stroke-width", 2);

showTooltip(d);

});

*// Add axes*

svg.selectAll(".axis")

.data(dimensions)

.enter().append("g")

.attr("class", "axis")

.attr("transform", d => `translate(${x(d)})`)

.each(function(d) { d3.select(this).call(d3.axisLeft(y[d])); })

.append("text")

.style("text-anchor", "middle")

.attr("y", -15)

.text(d => d)

.style("fill", "black");

*// Add brushing*

const brush = d3.brushY()

.extent([[-10, 0], [10, height]])

.on("brush", brushed);

svg.selectAll(".brush")

.data(dimensions)

.enter().append("g")

.attr("class", "brush")

.attr("transform", d => `translate(${x(d)})`)

.call(brush)

.call(brush.move, [0, height].map(y[dimensions[0]]));

function brushed(event, dimension) {

if (!event.sourceEvent) return;

const extent = event.selection.map(y[dimension].invert);

paths.style("opacity", d =>

extent[1] <= d[dimension] && d[dimension] <= extent[0] ? 1 : 0.1

);

}

}

**Optimization Tips:**

* Use Web Workers for data normalization with large datasets
* Implement level-of-detail rendering when >1000 claims
* Add a quadtree for faster mouseover detection

2. Interactive Sankey Diagram for Assessor vs Model Decisions

**Best for:** Visualizing claim decision flows and disagreements

javascript

function createSankey(selector, decisionData) {

*// Process decision paths*

const nodes = [...new Set([

...decisionData.map(d => `Model: ${d.model\_decision}`),

...decisionData.map(d => `Assessor: ${d.assessor\_decision}`),

...decisionData.map(d => `Final: ${d.final\_decision}`)

])].map(name => ({name}));

const links = decisionData.reduce((acc, claim) => {

acc.push({

source: `Model: ${claim.model\_decision}`,

target: `Assessor: ${claim.assessor\_decision}`,

value: claim.billed\_amount,

claim: claim.claim\_id

});

acc.push({

source: `Assessor: ${claim.assessor\_decision}`,

target: `Final: ${claim.final\_decision}`,

value: claim.billed\_amount,

claim: claim.claim\_id

});

return acc;

}, []);

*// Set up Sankey layout*

const sankey = d3.sankey()

.nodeWidth(15)

.nodePadding(10)

.extent([[1, 1], [width - 1, height - 6]]);

const {nodes: sankeyNodes, links: sankeyLinks} = sankey({

nodes: nodes.map(d => Object.assign({}, d)),

links: links.map(d => Object.assign({}, d))

});

*// Create SVG*

const svg = d3.create("svg")

.attr("viewBox", [0, 0, width, height]);

*// Add links*

const link = svg.append("g")

.selectAll("path")

.data(sankeyLinks)

.join("path")

.attr("d", d3.sankeyLinkHorizontal())

.attr("stroke", d => {

const source = d.source.name.split(": ")[1];

const target = d.target.name.split(": ")[1];

return source === target ? "#4CAF50" : "#F44336";

})

.attr("stroke-width", d => Math.max(1, d.width))

.on("mouseover", showLinkDetails);

*// Add nodes*

const node = svg.append("g")

.selectAll("rect")

.data(sankeyNodes)

.join("rect")

.attr("x", d => d.x0)

.attr("y", d => d.y0)

.attr("height", d => d.y1 - d.y0)

.attr("width", d => d.x1 - d.x0)

.attr("fill", d => {

const type = d.name.split(": ")[0];

return type === "Model" ? "#2196F3" :

type === "Assessor" ? "#FFC107" : "#9C27B0";

});

*// Add labels*

node.append("title").text(d => `${d.name}\n${d.value}`);

function showLinkDetails(event, d) {

const claims = decisionData.filter(

claim => claim.claim\_id === d.claim

);

*// Show detailed claim info in tooltip or side panel*

}

}

**Performance Enhancement:**

* Aggregate similar decision paths before rendering
* Use canvas rendering for >5000 claims
* Implement virtual scrolling for vertical space

3. Geospatial Provider Risk Map with D3 + Leaflet

**Best for:** Visualizing high-risk providers geographically

javascript

function createProviderMap(selector, providers) {

*// Set up Leaflet base map*

const map = L.map(selector).setView([37.8, -96], 4);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png').addTo(map);

*// Create D3 overlay*

const svg = d3.select(map.getPanes().overlayPane)

.append("svg")

.attr("width", "100%")

.attr("height", "100%");

const g = svg.append("g").attr("class", "leaflet-zoom-hide");

*// Projection function*

function projectPoint(x, y) {

const point = map.latLngToLayerPoint(new L.LatLng(y, x));

this.stream.point(point.x, point.y);

}

const transform = d3.geoTransform({point: projectPoint});

const path = d3.geoPath().projection(transform);

*// Create risk heatmap*

const riskScale = d3.scaleLinear()

.domain([0, d3.max(providers, d => d.risk\_score)])

.range(["#00E676", "#FF3D00"]);

const providerPoints = g.selectAll("circle")

.data(providers)

.enter().append("circle")

.attr("r", d => Math.sqrt(d.total\_claims) / 2)

.attr("fill", d => riskScale(d.risk\_score))

.attr("stroke", "#fff")

.attr("stroke-width", 1)

.on("mouseover", showProviderDetails);

*// Update on map move/zoom*

function update() {

providerPoints.attr("transform", d => {

const point = map.latLngToLayerPoint(new L.LatLng(d.lat, d.lng));

return `translate(${point.x},${point.y})`;

});

}

map.on("moveend", update);

*// Cluster markers at high zoom levels*

function clusterMarkers() {

if (map.getZoom() < 8) {

*// Implement marker clustering algorithm*

} else {

*// Show individual providers*

}

update();

}

map.on("zoomend", clusterMarkers);

}

**Advanced Features to Add:**

* Hexbin clustering for dense urban areas
* Voronoi polygons for provider service areas
* Animated flow lines showing patient referrals
* Time slider for temporal pattern analysis

4. Real-time Performance Optimization Techniques

For handling large healthcare datasets:

1. **Data Sampling & Aggregation**:

javascript

function aggregateClaims(claims, level) {

*// Sample implementation - aggregate by provider specialty*

return d3.rollup(claims,

v => ({

total\_claims: v.length,

total\_billed: d3.sum(v, d => d.billed\_amount),

avg\_risk: d3.mean(v, d => d.risk\_score)

}),

d => d[level] *// Group by specialty, region, etc*

);

}

1. **Web Worker Pipeline**:

javascript

*// main.js*

const worker = new Worker('data-processor.js');

worker.postMessage({command: 'process', data: rawClaims});

worker.onmessage = function(e) {

if (e.data.type === 'processed') {

updateVisualizations(e.data.results);

}

};

*// data-processor.js*

self.onmessage = function(e) {

if (e.data.command === 'process') {

const results = heavyProcessing(e.data.data);

self.postMessage({type: 'processed', results});

}

};

function heavyProcessing(data) {

*// Perform CPU-intensive operations*

}

1. **Canvas-based Rendering for Large Datasets**:

javascript

function renderScatterplotCanvas(data) {

const canvas = d3.select("#chart").append("canvas")

.attr("width", width)

.attr("height", height);

const context = canvas.node().getContext("2d");

*// Batch rendering*

const batchSize = 1000;

for (let i = 0; i < data.length; i += batchSize) {

const batch = data.slice(i, i + batchSize);

context.beginPath();

batch.forEach(d => {

context.moveTo(x(d.x), y(d.y));

context.arc(x(d.x), y(d.y), 2, 0, 2 \* Math.PI);

});

context.fillStyle = "steelblue";

context.fill();

}

}

5. Dynamic Threshold Simulation View

javascript

function createThresholdSimulator(data) {

const svg = d3.select("#simulator");

const controls = d3.select("#controls");

*// Initial parameters*

let params = {

threshold: 0.95,

sensitivity: 0.8,

minBilled: 1000

};

*// Update function*

function update() {

*// Filter data based on current parameters*

const filtered = data.filter(d =>

d.risk\_score >= quantile(data.map(d => d.risk\_score), params.threshold) &&

d.billed\_amount >= params.minBilled

);

*// Update visualizations*

renderHistogram(filtered);

renderSavingsProjection(filtered);

}

*// Interactive controls*

controls.selectAll(".slider")

.data(Object.keys(params))

.enter().append("input")

.attr("type", "range")

.attr("min", 0)

.attr("max", 1)

.attr("step", 0.01)

.attr("value", d => params[d])

.on("input", function(event, key) {

params[key] = +event.target.value;

update();

});

*// Initial render*

update();

}

Would you like me to provide implementations for any specific additional visualizations or focus on particular aspects like real-time updates, accessibility features, or mobile responsiveness?

Can you pls generate full code for dashboards

Complete FWA Dashboard Implementation with D3.js

I'll provide a comprehensive implementation for a Fraud, Waste, and Abuse (FWA) dashboard with multiple interconnected visualizations. This solution includes modular components, real-time data handling, and responsive design.

1. Main Dashboard Structure

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Healthcare FWA Analytics Dashboard</title>

<link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

<style>

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

margin: 0;

padding: 0;

background-color: #f5f7fa;

}

.dashboard {

display: grid;

grid-template-columns: 250px 1fr;

grid-template-rows: 60px 1fr;

height: 100vh;

}

.header {

grid-column: 1 / -1;

background: #2c3e50;

color: white;

padding: 0 20px;

display: flex;

align-items: center;

box-shadow: 0 2px 5px rgba(0,0,0,0.1);

}

.sidebar {

background: #34495e;

color: white;

padding: 20px;

overflow-y: auto;

}

.main-content {

display: grid;

grid-template-columns: repeat(12, 1fr);

grid-auto-rows: minmax(150px, auto);

gap: 20px;

padding: 20px;

overflow-y: auto;

}

.panel {

background: white;

border-radius: 5px;

box-shadow: 0 2px 10px rgba(0,0,0,0.05);

padding: 15px;

}

.panel-header {

font-weight: 600;

margin-bottom: 15px;

color: #2c3e50;

display: flex;

justify-content: space-between;

align-items: center;

}

.panel.full-width {

grid-column: 1 / -1;

}

.panel.half-width {

grid-column: span 6;

}

.panel.third-width {

grid-column: span 4;

}

.map-container {

height: 100%;

min-height: 400px;

}

.tooltip {

position: absolute;

padding: 10px;

background: rgba(0, 0, 0, 0.8);

color: white;

border-radius: 3px;

pointer-events: none;

font-size: 12px;

}

.filter-control {

margin-bottom: 15px;

}

.filter-control label {

display: block;

margin-bottom: 5px;

font-size: 14px;

}

select, input[type="range"] {

width: 100%;

}

.legend {

font-size: 12px;

}

</style>

</head>

<body>

<div class="dashboard">

<div class="header">

<h1>Healthcare FWA Monitoring Dashboard</h1>

</div>

<div class="sidebar">

<div class="filter-control">

<label for="time-period">Time Period</label>

<select id="time-period">

<option value="7d">Last 7 Days</option>

<option value="30d" selected>Last 30 Days</option>

<option value="90d">Last 90 Days</option>

<option value="ytd">Year to Date</option>

</select>

</div>

<div class="filter-control">

<label for="risk-threshold">Risk Threshold</label>

<input type="range" id="risk-threshold" min="0" max="100" value="75">

<div class="legend">Current: <span id="threshold-value">75</span>%</div>

</div>

<div class="filter-control">

<label for="provider-type">Provider Type</label>

<select id="provider-type">

<option value="all">All Providers</option>

<option value="hospital">Hospitals</option>

<option value="physician">Physicians</option>

<option value="lab">Labs</option>

</select>

</div>

<div class="filter-control">

<button id="refresh-data">Refresh Data</button>

</div>

<div class="summary-stats">

<h3>Key Metrics</h3>

<div id="total-claims" class="metric">Total Claims: <span class="value">-</span></div>

<div id="flagged-claims" class="metric">Flagged Claims: <span class="value">-</span></div>

<div id="potential-savings" class="metric">Potential Savings: <span class="value">-</span></div>

</div>

</div>

<div class="main-content">

<div class="panel full-width" id="risk-trend">

<div class="panel-header">Risk Trend Analysis</div>

<div class="chart-container"></div>

</div>

<div class="panel half-width" id="kri-outliers">

<div class="panel-header">KRI Outliers</div>

<div class="chart-container"></div>

</div>

<div class="panel half-width" id="decision-comparison">

<div class="panel-header">Assessor vs Model Decisions</div>

<div class="chart-container"></div>

</div>

<div class="panel full-width" id="provider-map">

<div class="panel-header">Provider Risk Geography</div>

<div class="map-container"></div>

</div>

<div class="panel third-width" id="los-analysis">

<div class="panel-header">Length of Stay Outliers</div>

<div class="chart-container"></div>

</div>

<div class="panel third-width" id="savings-potential">

<div class="panel-header">Savings Potential</div>

<div class="chart-container"></div>

</div>

<div class="panel third-width" id="top-providers">

<div class="panel-header">High-Risk Providers</div>

<div class="chart-container"></div>

</div>

<div class="panel full-width" id="simulation-view">

<div class="panel-header">Threshold Simulation</div>

<div class="chart-container"></div>

</div>

</div>

</div>

<div class="tooltip" id="tooltip"></div>

<script src="https://d3js.org/d3.v7.min.js"></script>

<script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>

<script src="https://d3js.org/d3-sankey.min.js"></script>

<script src="dashboard.js"></script>

</body>

</html>

2. JavaScript Implementation (dashboard.js)

javascript

*// Main Dashboard Controller*

class FWADashboard {

constructor() {

this.data = null;

this.filters = {

timePeriod: '30d',

riskThreshold: 75,

providerType: 'all'

};

this.initControls();

this.loadData();

}

initControls() {

*// Time period filter*

d3.select('#time-period').on('change', () => {

this.filters.timePeriod = d3.select('#time-period').node().value;

this.updateDashboard();

});

*// Risk threshold filter*

d3.select('#risk-threshold').on('input', () => {

const value = d3.select('#risk-threshold').node().value;

this.filters.riskThreshold = value;

d3.select('#threshold-value').text(value);

this.updateDashboard();

});

*// Provider type filter*

d3.select('#provider-type').on('change', () => {

this.filters.providerType = d3.select('#provider-type').node().value;

this.updateDashboard();

});

*// Refresh button*

d3.select('#refresh-data').on('click', () => this.loadData());

}

async loadData() {

try {

*// In a real implementation, this would fetch from an API*

*// For demo purposes, we'll use sample data*

const response = await fetch('fwa-data.json');

this.data = await response.json();

this.processData();

this.renderDashboard();

} catch (error) {

console.error('Error loading data:', error);

alert('Failed to load dashboard data');

}

}

processData() {

*// Apply time period filter*

const now = new Date();

let cutoffDate;

switch(this.filters.timePeriod) {

case '7d': cutoffDate = new Date(now.setDate(now.getDate() - 7)); break;

case '30d': cutoffDate = new Date(now.setDate(now.getDate() - 30)); break;

case '90d': cutoffDate = new Date(now.setDate(now.getDate() - 90)); break;

case 'ytd':

cutoffDate = new Date(now.getFullYear(), 0, 1);

break;

default: cutoffDate = new Date(0);

}

this.filteredData = this.data.claims.filter(claim =>

new Date(claim.service\_date) >= cutoffDate

);

*// Apply provider type filter*

if (this.filters.providerType !== 'all') {

this.filteredData = this.filteredData.filter(claim =>

claim.provider\_type === this.filters.providerType

);

}

*// Calculate risk threshold*

const riskScores = this.filteredData.map(d => d.risk\_score);

this.riskCutoff = d3.quantile(riskScores, this.filters.riskThreshold / 100);

*// Update summary stats*

d3.select('#total-claims .value').text(this.filteredData.length.toLocaleString());

const flagged = this.filteredData.filter(d => d.risk\_score >= this.riskCutoff).length;

d3.select('#flagged-claims .value').text(flagged.toLocaleString());

const savings = d3.sum(

this.filteredData.filter(d => d.risk\_score >= this.riskCutoff),

d => d.billed\_amount - d.allowed\_amount

);

d3.select('#potential-savings .value').text(`$${savings.toLocaleString()}`);

}

renderDashboard() {

this.renderRiskTrend();

this.renderKRIOutliers();

this.renderDecisionComparison();

this.renderProviderMap();

this.renderLOSAnalysis();

this.renderSavingsPotential();

this.renderTopProviders();

this.renderSimulationView();

}

updateDashboard() {

this.processData();

this.renderDashboard();

}

*// Visualization Methods*

renderRiskTrend() {

const container = d3.select('#risk-trend .chart-container');

container.selectAll('\*').remove();

*// Group data by date*

const timeFormat = d3.timeFormat('%Y-%m-%d');

const nestedData = d3.rollup(

this.filteredData,

v => ({

total: v.length,

flagged: v.filter(d => d.risk\_score >= this.riskCutoff).length,

avgRisk: d3.mean(v, d => d.risk\_score)

}),

d => timeFormat(new Date(d.service\_date))

);

const trendData = Array.from(nestedData, ([date, values]) => ({

date: new Date(date),

...values

})).sort((a, b) => a.date - b.date);

const margin = {top: 20, right: 30, bottom: 40, left: 50};

const width = container.node().clientWidth - margin.left - margin.right;

const height = 300 - margin.top - margin.bottom;

const svg = container.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})`);

*// Scales*

const x = d3.scaleTime()

.domain(d3.extent(trendData, d => d.date))

.range([0, width]);

const y = d3.scaleLinear()

.domain([0, d3.max(trendData, d => d.flagged)])

.range([height, 0]);

*// Line generator*

const line = d3.line()

.x(d => x(d.date))

.y(d => y(d.flagged));

*// Add line*

svg.append('path')

.datum(trendData)

.attr('fill', 'none')

.attr('stroke', '#e74c3c')

.attr('stroke-width', 2)

.attr('d', line);

*// Add axes*

svg.append('g')

.attr('transform', `translate(0,${height})`)

.call(d3.axisBottom(x));

svg.append('g')

.call(d3.axisLeft(y));

*// Add title*

svg.append('text')

.attr('x', width / 2)

.attr('y', -10)

.attr('text-anchor', 'middle')

.text('Flagged Claims Over Time');

}

renderKRIOutliers() {

const container = d3.select('#kri-outliers .chart-container');

container.selectAll('\*').remove();

*// Get KRI metrics from first claim (simplified)*

const kris = Object.keys(this.filteredData[0].kri\_scores || {

'LOS Score': 0,

'Cost Anomaly': 0,

'Frequency': 0,

'Clinical Mismatch': 0

};

const margin = {top: 20, right: 20, bottom: 40, left: 40};

const width = container.node().clientWidth - margin.left - margin.right;

const height = 300 - margin.top - margin.bottom;

const svg = container.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})`);

*// Calculate outlier counts per KRI*

const kriData = Object.keys(kris).map(kri => {

const values = this.filteredData.map(d => d.kri\_scores[kri]);

const cutoff = d3.quantile(values, 0.95); *// Using 95th percentile as outlier threshold*

return {

name: kri,

outliers: values.filter(v => v >= cutoff).length,

total: values.length,

rate: values.filter(v => v >= cutoff).length / values.length

};

});

*// Scales*

const x = d3.scaleBand()

.domain(kriData.map(d => d.name))

.range([0, width])

.padding(0.2);

const y = d3.scaleLinear()

.domain([0, d3.max(kriData, d => d.rate)])

.range([height, 0]);

*// Add bars*

svg.selectAll('.bar')

.data(kriData)

.enter().append('rect')

.attr('class', 'bar')

.attr('x', d => x(d.name))

.attr('y', d => y(d.rate))

.attr('width', x.bandwidth())

.attr('height', d => height - y(d.rate))

.attr('fill', d => d.rate > 0.1 ? '#e74c3c' : '#3498db');

*// Add axes*

svg.append('g')

.attr('transform', `translate(0,${height})`)

.call(d3.axisBottom(x));

svg.append('g')

.call(d3.axisLeft(y).ticks(5).tickFormat(d3.format('.0%')));

*// Add value labels*

svg.selectAll('.label')

.data(kriData)

.enter().append('text')

.attr('x', d => x(d.name) + x.bandwidth() / 2)

.attr('y', d => y(d.rate) - 5)

.attr('text-anchor', 'middle')

.text(d => d3.format('.1%')(d.rate));

}

renderDecisionComparison() {

const container = d3.select('#decision-comparison .chart-container');

container.selectAll('\*').remove();

const margin = {top: 20, right: 20, bottom: 60, left: 40};

const width = container.node().clientWidth - margin.left - margin.right;

const height = 300 - margin.top - margin.bottom;

const svg = container.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})`);

*// Prepare data - count decisions*

const decisions = ['Approved', 'Denied', 'Pending'];

const modelData = decisions.map(decision => ({

decision,

count: this.filteredData.filter(d => d.model\_decision === decision).length,

type: 'Model'

}));

const assessorData = decisions.map(decision => ({

decision,

count: this.filteredData.filter(d => d.assessor\_decision === decision).length,

type: 'Assessor'

}));

const combinedData = [...modelData, ...assessorData];

*// Scales*

const x0 = d3.scaleBand()

.domain(decisions)

.range([0, width])

.paddingInner(0.1);

const x1 = d3.scaleBand()

.domain(['Model', 'Assessor'])

.range([0, x0.bandwidth()])

.padding(0.05);

const y = d3.scaleLinear()

.domain([0, d3.max(combinedData, d => d.count)])

.range([height, 0]);

*// Color scale*

const color = d3.scaleOrdinal()

.domain(['Model', 'Assessor'])

.range(['#3498db', '#2ecc71']);

*// Add groups for each decision type*

const decisionGroups = svg.selectAll('.decision-group')

.data(decisions)

.enter().append('g')

.attr('class', 'decision-group')

.attr('transform', d => `translate(${x0(d)},0)`);

*// Add bars*

decisionGroups.selectAll('rect')

.data(d => {

return ['Model', 'Assessor'].map(type => ({

decision: d,

type,

count: combinedData.find(item => item.decision === d && item.type === type).count

}));

})

.enter().append('rect')

.attr('x', d => x1(d.type))

.attr('y', d => y(d.count))

.attr('width', x1.bandwidth())

.attr('height', d => height - y(d.count))

.attr('fill', d => color(d.type));

*// Add axes*

svg.append('g')

.attr('transform', `translate(0,${height})`)

.call(d3.axisBottom(x0));

svg.append('g')

.call(d3.axisLeft(y));

*// Add legend*

const legend = svg.append('g')

.attr('transform', `translate(${width / 2 - 50},${height + 30})`);

['Model', 'Assessor'].forEach((type, i) => {

legend.append('rect')

.attr('x', i \* 60)

.attr('y', 0)

.attr('width', 10)

.attr('height', 10)

.attr('fill', color(type));

legend.append('text')

.attr('x', i \* 60 + 15)

.attr('y', 10)

.text(type)

.style('font-size', '12px');

});

}

renderProviderMap() {

const container = d3.select('#provider-map .map-container');

container.selectAll('\*').remove();

*// Initialize Leaflet map*

const map = L.map(container.node()).setView([37.8, -96], 4);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {

attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'

}).addTo(map);

*// Create D3 overlay*

const svg = d3.select(map.getPanes().overlayPane).append('svg');

const g = svg.append('g').attr('class', 'leaflet-zoom-hide');

*// Process provider data*

const providerData = this.data.providers.filter(provider => {

const hasClaims = this.filteredData.some(claim => claim.provider\_id === provider.id);

return hasClaims;

});

*// Create risk scale*

const riskScale = d3.scaleLinear()

.domain([0, d3.max(providerData, d => d.risk\_score)])

.range(['#2ecc71', '#e74c3c']);

*// Draw markers*

const providerMarkers = g.selectAll('circle')

.data(providerData)

.enter().append('circle')

.attr('r', d => Math.sqrt(d.claim\_count) / 2)

.attr('fill', d => riskScale(d.risk\_score))

.attr('stroke', '#fff')

.attr('stroke-width', 1)

.on('mouseover', (event, d) => {

const tooltip = d3.select('#tooltip');

tooltip.style('opacity', 1)

.html(`

<strong>${d.name}</strong><br>

Type: ${d.type}<br>

Risk Score: ${d.risk\_score.toFixed(1)}<br>

Claims: ${d.claim\_count}<br>

Location: ${d.city}, ${d.state}

`)

.style('left', (event.pageX + 10) + 'px')

.style('top', (event.pageY - 28) + 'px');

})

.on('mouseout', () => {

d3.select('#tooltip').style('opacity', 0);

});

*// Update function for map movement*

function update() {

providerMarkers.attr('transform', d => {

const point = map.latLngToLayerPoint([d.lat, d.lng]);

return `translate(${point.x},${point.y})`;

});

}

map.on('moveend', update);

update();

}

renderLOSAnalysis() {

const container = d3.select('#los-analysis .chart-container');

container.selectAll('\*').remove();

*// Filter to inpatient claims with LOS*

const losData = this.filteredData.filter(d => d.los\_days != null);

if (losData.length === 0) {

container.append('div').text('No LOS data available for selected filters');

return;

}

const margin = {top: 20, right: 20, bottom: 40, left: 40};

const width = container.node().clientWidth - margin.left - margin.right;

const height = 300 - margin.top - margin.bottom;

const svg = container.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})`);

*// Calculate outliers (beyond 1.5 \* IQR)*

const values = losData.map(d => d.los\_days).sort(d3.ascending);

const q1 = d3.quantile(values, 0.25);

const q3 = d3.quantile(values, 0.75);

const iqr = q3 - q1;

const lowerBound = q1 - 1.5 \* iqr;

const upperBound = q3 + 1.5 \* iqr;

*// Prepare box plot data*

const boxData = {

min: d3.min(values),

q1,

median: d3.quantile(values, 0.5),

q3,

max: d3.max(values),

outliers: values.filter(v => v < lowerBound || v > upperBound)

};

*// Scales*

const x = d3.scaleBand()

.domain(['LOS'])

.range([0, width])

.padding(0.5);

const y = d3.scaleLinear()

.domain([0, d3.max(values)])

.range([height, 0]);

*// Draw main box*

svg.append('rect')

.attr('x', x('LOS') + x.bandwidth() \* 0.25)

.attr('y', y(boxData.q3))

.attr('width', x.bandwidth() \* 0.5)

.attr('height', y(boxData.q1) - y(boxData.q3))

.attr('fill', '#3498db')

.attr('stroke', '#2980b9');

*// Draw median line*

svg.append('line')

.attr('x1', x('LOS') + x.bandwidth() \* 0.25)

.attr('x2', x('LOS') + x.bandwidth() \* 0.75)

.attr('y1', y(boxData.median))

.attr('y2', y(boxData.median))

.attr('stroke', '#fff')

.attr('stroke-width', 2);

*// Draw whiskers*

svg.append('line')

.attr('x1', x('LOS') + x.bandwidth() / 2)

.attr('x2', x('LOS') + x.bandwidth() / 2)

.attr('y1', y(boxData.min))

.attr('y2', y(boxData.q1))

.attr('stroke', '#2980b9')

.attr('stroke-width', 1.5);

svg.append('line')

.attr('x1', x('LOS') + x.bandwidth() / 2)

.attr('x2', x('LOS') + x.bandwidth() / 2)

.attr('y1', y(boxData.q3))

.attr('y2', y(boxData.max))

.attr('stroke', '#2980b9')

.attr('stroke-width', 1.5);

*// Draw outliers*

svg.selectAll('.outlier')

.data(boxData.outliers)

.enter().append('circle')

.attr('class', 'outlier')

.attr('cx', x('LOS') + x.bandwidth() / 2)

.attr('cy', d => y(d))

.attr('r', 3)

.attr('fill', '#e74c3c')

.attr('stroke', '#c0392b');

*// Add axes*

svg.append('g')

.attr('transform', `translate(0,${height})`)

.call(d3.axisBottom(x));

svg.append('g')

.call(d3.axisLeft(y));

*// Add title*

svg.append('text')

.attr('x', width / 2)

.attr('y', -10)

.attr('text-anchor', 'middle')

.text('Length of Stay Distribution');

}

renderSavingsPotential() {

const container = d3.select('#savings-potential .chart-container');

container.selectAll('\*').remove();

*// Calculate savings by category*

const flaggedClaims = this.filteredData.filter(d => d.risk\_score >= this.riskCutoff);

const savingsByCategory = d3.rollup(

flaggedClaims,

v => d3.sum(v, d => d.billed\_amount - d.allowed\_amount),

d => d.service\_category

);

const savingsData = Array.from(savingsByCategory, ([category, savings]) => ({

category,

savings

})).sort((a, b) => b.savings - a.savings);

const margin = {top: 20, right: 20, bottom: 40, left: 60};

const width = container.node().clientWidth - margin.left - margin.right;

const height = 300 - margin.top - margin.bottom;

const svg = container.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})`);

*// Scales*

const x = d3.scaleLinear()

.domain([0, d3.max(savingsData, d => d.savings)])

.range([0, width]);

const y = d3.scaleBand()

.domain(savingsData.map(d => d.category))

.range([0, height])

.padding(0.2);

*// Add bars*

svg.selectAll('.bar')

.data(savingsData)

.enter().append('rect')

.attr('class', 'bar')

.attr('y', d => y(d.category))

.attr('height', y.bandwidth())

.attr('x', 0)

.attr('width', d => x(d.savings))

.attr('fill', '#9b59b6');

*// Add value labels*

svg.selectAll('.label')

.data(savingsData)

.enter().append('text')

.attr('y', d => y(d.category) + y.bandwidth() / 2)

.attr('x', d => x(d.savings) + 5)

.attr('dy', '0.35em')

.text(d => `$${d3.format(',.0f')(d.savings)}`)

.style('font-size', '10px')

.style('fill', '#fff');

*// Add axes*

svg.append('g')

.attr('transform', `translate(0,${height})`)

.call(d3.axisBottom(x).tickFormat(d => `$${d3.format(',.0f')(d)}`));

svg.append('g')

.call(d3.axisLeft(y));

}

renderTopProviders() {

const container = d3.select('#top-providers .chart-container');

container.selectAll('\*').remove();

*// Group claims by provider and calculate risk metrics*

const providerClaims = d3.rollup(

this.filteredData,

v => ({

count: v.length,

totalBilled: d3.sum(v, d => d.billed\_amount),

avgRisk: d3.mean(v, d => d.risk\_score),

flagged: v.filter(d => d.risk\_score >= this.riskCutoff).length

}),

d => d.provider\_id

);

*// Merge with provider details*

const providerData = Array.from(providerClaims, ([id, metrics]) => {

const provider = this.data.providers.find(p => p.id === id) || {

name: 'Unknown Provider',

type: 'Unknown',

city: '',

state: ''

};

return {

...provider,

...metrics,

flaggedRate: metrics.flagged / metrics.count

};

})

.sort((a, b) => b.flaggedRate - a.flaggedRate)

.slice(0, 5); *// Top 5*

if (providerData.length === 0) {

container.append('div').text('No provider data available');

return;

}

const margin = {top: 20, right: 20, bottom: 40, left: 100};

const width = container.node().clientWidth - margin.left - margin.right;

const height = 300 - margin.top - margin.bottom;

const svg = container.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})`);

*// Scales*

const x = d3.scaleLinear()

.domain([0, 1])

.range([0, width]);

const y = d3.scaleBand()

.domain(providerData.map(d => d.name))

.range([0, height])

.padding(0.2);

*// Add bars*

svg.selectAll('.bar')

.data(providerData)

.enter().append('rect')

.attr('class', 'bar')

.attr('y', d => y(d.name))

.attr('height', y.bandwidth())

.attr('x', 0)

.attr('width', d => x(d.flaggedRate))

.attr('fill', d => d.flaggedRate > 0.3 ? '#e74c3c' : '#f39c12');

*// Add value labels*

svg.selectAll('.label')

.data(providerData)

.enter().append('text')

.attr('y', d => y(d.name) + y.bandwidth() / 2)

.attr('x', d => x(d.flaggedRate) + 5)

.attr('dy', '0.35em')

.text(d => d3.format('.0%')(d.flaggedRate))

.style('font-size', '10px')

.style('fill', '#fff');

*// Add axes*

svg.append('g')

.attr('transform', `translate(0,${height})`)

.call(d3.axisBottom(x).tickFormat(d3.format('.0%')));

svg.append('g')

.call(d3.axisLeft(y));

*// Add title*

svg.append('text')

.attr('x', width / 2)

.attr('y', -10)

.attr('text-anchor', 'middle')

.text('Top High-Risk Providers');

}

renderSimulationView() {

const container = d3.select('#simulation-view .chart-container');

container.selectAll('\*').remove();

const margin = {top: 20, right: 30, bottom: 50, left: 60};

const width = container.node().clientWidth - margin.left - margin.right;

const height = 300 - margin.top - margin.bottom;

const svg = container.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})`);

*// Prepare data for different threshold levels*

const thresholdLevels = d3.range(50, 100, 5); *// From 50% to 95% in 5% steps*

const simulationData = thresholdLevels.map(threshold => {

const cutoff = d3.quantile(

this.filteredData.map(d => d.risk\_score),

threshold / 100

);

const flaggedClaims = this.filteredData.filter(d => d.risk\_score >= cutoff);

return {

threshold,

claims: flaggedClaims.length,

savings: d3.sum(flaggedClaims, d => d.billed\_amount - d.allowed\_amount),

precision: flaggedClaims.length > 0 ?

flaggedClaims.filter(d => d.fraud\_flag).length / flaggedClaims.length : 0

};

});

*// Scales*

const x = d3.scaleLinear()

.domain([50, 95])

.range([0, width]);

const y = d3.scaleLinear()

.domain([0, d3.max(simulationData, d => d.savings)])

.range([height, 0]);

*// Line generator*

const line = d3.line()

.x(d => x(d.threshold))

.y(d => y(d.savings));

*// Add line*

svg.append('path')

.datum(simulationData)

.attr('fill', 'none')

.attr('stroke', '#3498db')

.attr('stroke-width', 2)

.attr('d', line);

*// Add circles*

svg.selectAll('.point')

.data(simulationData)

.enter().append('circle')

.attr('class', 'point')

.attr('cx', d => x(d.threshold))

.attr('cy', d => y(d.savings))

.attr('r', 4)

.attr('fill', '#3498db')

.on('mouseover', (event, d) => {

const tooltip = d3.select('#tooltip');

tooltip.style('opacity', 1)

.html(`

<strong>Threshold: ${d.threshold}%</strong><br>

Flagged Claims: ${d.claims}<br>

Potential Savings: $${d3.format(',.0f')(d.savings)}<br>

Estimated Precision: ${d3.format('.1%')(d.precision)}

`)

.style('left', (event.pageX + 10) + 'px')

.style('top', (event.pageY - 28) + 'px');

})

.on('mouseout', () => {

d3.select('#tooltip').style('opacity', 0);

});

*// Add axes*

svg.append('g')

.attr('transform', `translate(0,${height})`)

.call(d3.axisBottom(x).tickFormat(d => `${d}%`));

svg.append('g')

.call(d3.axisLeft(y).tickFormat(d => `$${d3.format(',.0f')(d)}`));

*// Add title*

svg.append('text')

.attr('x', width / 2)

.attr('y', -10)

.attr('text-anchor', 'middle')

.text('Savings Potential by Risk Threshold');

*// Add current threshold marker*

svg.append('line')

.attr('x1', x(this.filters.riskThreshold))

.attr('x2', x(this.filters.riskThreshold))

.attr('y1', 0)

.attr('y2', height)

.attr('stroke', '#e74c3c')

.attr('stroke-dasharray', '3,3')

.attr('stroke-width', 1);

}

}

*// Initialize dashboard when DOM is loaded*

document.addEventListener('DOMContentLoaded', () => {

const dashboard = new FWADashboard();

});

3. Sample Data Structure (fwa-data.json)

json

{

"claims": [

{

"claim\_id": "CLM1001",

"member\_id": "M1001",

"provider\_id": "P1001",

"service\_date": "2023-01-15",

"service\_category": "Inpatient",

"procedure\_code": "DRG001",

"diagnosis\_code": "ICD10-A01",

"billed\_amount": 12500,

"allowed\_amount": 9500,

"los\_days": 4,

"model\_decision": "Approved",

"assessor\_decision": "Approved",

"final\_decision": "Approved",

"fraud\_flag": false,

"risk\_score": 0.35,

"kri\_scores": {

"LOS Score": 0.4,

"Cost Anomaly": 0.3,

"Frequency": 0.2,

"Clinical Mismatch": 0.1

}

},

{

"claim\_id": "CLM1002",

"member\_id": "M1002",

"provider\_id": "P1002",

"service\_date": "2023-01-18",

"service\_category": "Outpatient",

"procedure\_code": "CPT99214",

"diagnosis\_code": "ICD10-E11",

"billed\_amount": 350,

"allowed\_amount": 250,

"model\_decision": "Denied",

"assessor\_decision": "Approved",

"final\_decision": "Pending",

"fraud\_flag": true,

"risk\_score": 0.85,

"kri\_scores": {

"LOS Score": 0.1,

"Cost Anomaly": 0.9,

"Frequency": 0.8,

"Clinical Mismatch": 0.7

}

}

],

"providers": [

{

"id": "P1001",

"name": "City General Hospital",

"type": "hospital",

"address": "123 Main St",

"city": "Metropolis",

"state": "NY",

"zip": "10001",

"lat": 40.7128,

"lng": -74.0060,

"risk\_score": 0.4,

"claim\_count": 1250

},

{

"id": "P1002",

"name": "Downtown Clinic",

"type": "clinic",

"address": "456 Oak Ave",

"city": "Metropolis",

"state": "NY",

"zip": "10002",

"lat": 40.7100,

"lng": -74.0080,

"risk\_score": 0.8,

"claim\_count": 850

}

]

}

4. Key Features Implemented

1. **Interactive Filtering**:
   * Time period selection (7d, 30d, 90d, YTD)
   * Dynamic risk threshold adjustment
   * Provider type filtering
2. **Comprehensive Visualizations**:
   * Time series trend analysis
   * KRI outlier detection
   * Decision comparison charts
   * Geographic provider mapping
   * Length of stay analysis
   * Savings potential by category
   * High-risk provider identification
   * Threshold simulation
3. **Performance Optimizations**:
   * Data aggregation for large datasets
   * Responsive design for all components
   * Efficient data processing pipelines
4. **User Experience**:
   * Tooltips with detailed information
   * Consistent color coding
   * Clear labeling and legends
   * Interactive elements

5. Deployment Instructions

1. Create three files:
   * index.html (copy from section 1)
   * dashboard.js (copy from section 2)
   * fwa-data.json (copy from section 3)
2. Serve the files using a local web server:

bash

python -m http.server 8000

1. Open http://localhost:8000 in your browser

6. Enhancement Opportunities

1. **Real Data Integration**:
   * Connect to a backend API instead of static JSON
   * Implement WebSocket for real-time updates
2. **Advanced Analytics**:
   * Add predictive modeling visualizations
   * Implement anomaly detection algorithms
3. **User Management**:
   * Add login/authentication
   * Role-based view customization
4. **Export Capabilities**:
   * PDF/PNG export of dashboards
   * Data export in CSV/Excel formats

Would you like me to elaborate on any specific component or add additional features to this implementation?

Comprehensive FWA Dashboard Project with D3.js

I'll provide a complete project structure with separate dashboards for each FWA analysis section, featuring advanced D3.js visualizations. This implementation includes modular components, interactive features, and a clean architecture.

Project Structure

text

fwa-dashboard/

├── index.html # Main entry point

├── css/

│ └── styles.css # Global styles

├── js/

│ ├── main.js # App controller

│ ├── utils/ # Utility functions

│ │ ├── dataLoader.js # Data loading/processing

│ │ └── helpers.js # Visualization helpers

│ └── dashboards/ # Individual dashboard modules

│ ├── kriOutliers.js

│ ├── decisionComparison.js

│ ├── providerWatchlist.js

│ ├── memberLens.js

│ ├── lengthOfStay.js

│ ├── savingsPotential.js

│ ├── clinicalAttributes.js

│ ├── trendAnalysis.js

│ └── simulationView.js

├── data/ # Sample data files

│ └── fwa-data.json

└── assets/ # Images/icons

1. Main HTML File (index.html)

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Healthcare FWA Analytics</title>

<link rel="stylesheet" href="css/styles.css">

<link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

</head>

<body>

<header>

<h1>Healthcare FWA Monitoring System</h1>

<nav id="dashboard-nav">

<ul>

<li data-dashboard="kri-outliers">KRI Outliers</li>

<li data-dashboard="decision-comparison">Decision Analysis</li>

<li data-dashboard="provider-watchlist">Provider Watchlist</li>

<li data-dashboard="length-of-stay">Length of Stay</li>

<li data-dashboard="savings-potential">Savings Potential</li>

<li data-dashboard="simulation-view">Simulation</li>

</ul>

</nav>

<div class="filters">

<select id="time-period">

<option value="7d">Last 7 Days</option>

<option value="30d" selected>Last 30 Days</option>

<option value="90d">Last 90 Days</option>

</select>

<input type="range" id="risk-threshold" min="50" max="99" value="85">

<span id="threshold-value">85%</span>

</div>

</header>

<main id="dashboard-container">

*<!-- Dashboards will be loaded here dynamically -->*

<div class="loading">Loading dashboard...</div>

</main>

<div id="tooltip" class="tooltip"></div>

<script src="https://d3js.org/d3.v7.min.js"></script>

<script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>

<script src="https://d3js.org/d3-sankey.min.js"></script>

<script src="https://d3js.org/d3-hexbin.v0.2.min.js"></script>

<script src="js/utils/dataLoader.js"></script>

<script src="js/utils/helpers.js"></script>

<script src="js/main.js"></script>

</body>

</html>

2. CSS Styles (css/styles.css)

css

:root {

--primary: #3498db;

--secondary: #2ecc71;

--danger: #e74c3c;

--warning: #f39c12;

--dark: #2c3e50;

--light: #ecf0f1;

}

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

margin: 0;

padding: 0;

background-color: #f8f9fa;

color: #333;

}

header {

background: var(--dark);

color: white;

padding: 1rem 2rem;

display: flex;

justify-content: space-between;

align-items: center;

box-shadow: 0 2px 10px rgba(0,0,0,0.1);

}

nav ul {

display: flex;

list-style: none;

margin: 0;

padding: 0;

}

nav li {

margin: 0 1rem;

padding: 0.5rem 1rem;

cursor: pointer;

border-radius: 4px;

transition: background 0.3s;

}

nav li:hover, nav li.active {

background: rgba(255,255,255,0.2);

}

.filters {

display: flex;

align-items: center;

gap: 1rem;

}

select, input[type="range"] {

padding: 0.5rem;

border-radius: 4px;

border: none;

}

#dashboard-container {

padding: 2rem;

min-height: 80vh;

}

.dashboard {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(600px, 1fr));

gap: 2rem;

}

.panel {

background: white;

border-radius: 8px;

box-shadow: 0 2px 15px rgba(0,0,0,0.05);

padding: 1.5rem;

}

.panel-header {

font-size: 1.2rem;

font-weight: 600;

margin-bottom: 1.5rem;

color: var(--dark);

border-bottom: 1px solid #eee;

padding-bottom: 0.5rem;

}

.tooltip {

position: absolute;

padding: 0.75rem;

background: rgba(0, 0, 0, 0.85);

color: white;

border-radius: 4px;

pointer-events: none;

font-size: 0.9rem;

max-width: 300px;

z-index: 100;

}

.loading {

display: flex;

justify-content: center;

align-items: center;

height: 300px;

font-size: 1.2rem;

color: #777;

}

*/\* Animation for visualizations \*/*

.path-transition {

transition: all 0.3s ease-in-out;

}

*/\* Responsive adjustments \*/*

@media (max-width: 1200px) {

header {

flex-direction: column;

align-items: flex-start;

}

nav ul {

margin-top: 1rem;

flex-wrap: wrap;

}

}

3. Main Application Controller (js/main.js)

javascript

class FWAApplication {

constructor() {

this.dashboards = {

'kri-outliers': null,

'decision-comparison': null,

'provider-watchlist': null,

'length-of-stay': null,

'savings-potential': null,

'simulation-view': null

};

this.currentDashboard = null;

this.data = null;

this.filters = {

timePeriod: '30d',

riskThreshold: 85

};

this.initEventListeners();

this.loadData();

}

initEventListeners() {

*// Dashboard navigation*

document.querySelectorAll('#dashboard-nav li').forEach(item => {

item.addEventListener('click', () => {

const dashboardId = item.getAttribute('data-dashboard');

this.loadDashboard(dashboardId);

*// Update active state*

document.querySelectorAll('#dashboard-nav li').forEach(li => {

li.classList.remove('active');

});

item.classList.add('active');

});

});

*// Time period filter*

document.getElementById('time-period').addEventListener('change', (e) => {

this.filters.timePeriod = e.target.value;

this.updateCurrentDashboard();

});

*// Risk threshold filter*

document.getElementById('risk-threshold').addEventListener('input', (e) => {

this.filters.riskThreshold = parseInt(e.target.value);

document.getElementById('threshold-value').textContent = `${e.target.value}%`;

this.updateCurrentDashboard();

});

}

async loadData() {

try {

*// Show loading state*

document.getElementById('dashboard-container').innerHTML =

'<div class="loading">Loading data...</div>';

*// Load sample data (in real app, this would be an API call)*

const response = await fetch('data/fwa-data.json');

this.data = await response.json();

*// Process data*

this.processData();

*// Load default dashboard*

this.loadDashboard('kri-outliers');

document.querySelector('#dashboard-nav li[data-dashboard="kri-outliers"]').classList.add('active');

} catch (error) {

console.error('Error loading data:', error);

document.getElementById('dashboard-container').innerHTML =

'<div class="error">Failed to load data. Please try again later.</div>';

}

}

processData() {

*// Apply time period filter*

const now = new Date();

let cutoffDate;

switch(this.filters.timePeriod) {

case '7d': cutoffDate = new Date(now.setDate(now.getDate() - 7)); break;

case '30d': cutoffDate = new Date(now.setDate(now.getDate() - 30)); break;

case '90d': cutoffDate = new Date(now.setDate(now.getDate() - 90)); break;

default: cutoffDate = new Date(0);

}

this.filteredData = {

claims: this.data.claims.filter(claim =>

new Date(claim.service\_date) >= cutoffDate

),

providers: this.data.providers

};

*// Calculate risk threshold*

const riskScores = this.filteredData.claims.map(d => d.risk\_score);

this.riskCutoff = d3.quantile(riskScores, this.filters.riskThreshold / 100);

}

async loadDashboard(dashboardId) {

*// Unload current dashboard*

if (this.currentDashboard && this.dashboards[this.currentDashboard]) {

this.dashboards[this.currentDashboard].unload();

}

*// Show loading state*

document.getElementById('dashboard-container').innerHTML =

'<div class="loading">Loading dashboard...</div>';

try {

*// Load dashboard module if not already loaded*

if (!this.dashboards[dashboardId]) {

const module = await import(`./dashboards/${dashboardId}.js`);

this.dashboards[dashboardId] = new module.default(

document.getElementById('dashboard-container'),

this.filteredData,

this.riskCutoff

);

}

*// Render dashboard*

this.dashboards[dashboardId].render();

this.currentDashboard = dashboardId;

} catch (error) {

console.error(`Error loading dashboard ${dashboardId}:`, error);

document.getElementById('dashboard-container').innerHTML =

'<div class="error">Failed to load dashboard. Please try again.</div>';

}

}

updateCurrentDashboard() {

if (!this.currentDashboard) return;

*// Reprocess data with current filters*

this.processData();

*// Update current dashboard*

if (this.dashboards[this.currentDashboard]) {

this.dashboards[this.currentDashboard].updateData(

this.filteredData,

this.riskCutoff

);

}

}

}

*// Initialize application when DOM is loaded*

document.addEventListener('DOMContentLoaded', () => {

const app = new FWAApplication();

});

4. Dashboard Modules

KRI Outliers Dashboard (js/dashboards/kriOutliers.js)

javascript

export default class KRIOutliersDashboard {

constructor(container, data, riskCutoff) {

this.container = container;

this.data = data;

this.riskCutoff = riskCutoff;

this.margin = {top: 40, right: 30, bottom: 60, left: 60};

this.width = 800 - this.margin.left - this.margin.right;

this.height = 500 - this.margin.top - this.margin.bottom;

}

render() {

this.container.innerHTML = `

<div class="dashboard">

<div class="panel">

<div class="panel-header">Key Risk Indicator Outliers</div>

<div class="chart-container" id="parallel-coordinates"></div>

</div>

<div class="panel">

<div class="panel-header">Outlier Distribution by KRI</div>

<div class="chart-container" id="outlier-distribution"></div>

</div>

</div>

`;

this.renderParallelCoordinates();

this.renderOutlierDistribution();

}

renderParallelCoordinates() {

const container = d3.select('#parallel-coordinates');

const svg = container.append('svg')

.attr('width', this.width + this.margin.left + this.margin.right)

.attr('height', this.height + this.margin.top + this.margin.bottom)

.append('g')

.attr('transform', `translate(${this.margin.left},${this.margin.top})`);

*// Get KRI dimensions from first claim*

const dimensions = Object.keys(this.data.claims[0].kri\_scores);

*// Prepare data - normalize values and add outlier flag*

const normalizedData = this.data.claims.map(claim => {

const normalized = { id: claim.claim\_id, outlier: claim.risk\_score >= this.riskCutoff };

dimensions.forEach(dim => {

normalized[dim] = claim.kri\_scores[dim];

});

return normalized;

});

*// Create scales for each dimension*

const y = {};

dimensions.forEach(dim => {

y[dim] = d3.scaleLinear()

.domain([0, 1])

.range([this.height, 0]);

});

const x = d3.scalePoint()

.domain(dimensions)

.range([0, this.width])

.padding(0.5);

*// Draw lines*

const line = d3.line()

.defined(d => !isNaN(d.value))

.x(d => x(d.name))

.y(d => y[d.name](d.value));

svg.selectAll('.path')

.data(normalizedData)

.enter().append('path')

.attr('class', 'path-transition')

.attr('d', d => line(dimensions.map(name => ({ name, value: d[name] }))))

.attr('stroke', d => d.outlier ? '#e74c3c' : '#3498db')

.attr('stroke-width', 1)

.attr('opacity', d => d.outlier ? 0.7 : 0.3)

.on('mouseover', this.showTooltip)

.on('mouseout', () => d3.select('#tooltip').style('opacity', 0));

*// Add axes*

svg.selectAll('.axis')

.data(dimensions)

.enter().append('g')

.attr('class', 'axis')

.attr('transform', d => `translate(${x(d)})`)

.each(function(d) {

d3.select(this).call(d3.axisLeft(y[d]).ticks(5));

})

.append('text')

.attr('y', -15)

.attr('text-anchor', 'middle')

.text(d => d)

.style('fill', 'var(--dark)');

*// Add title*

svg.append('text')

.attr('x', this.width / 2)

.attr('y', -10)

.attr('text-anchor', 'middle')

.style('font-size', '14px')

.style('font-weight', 'bold')

.text('Parallel Coordinates of KRI Scores');

}

renderOutlierDistribution() {

const container = d3.select('#outlier-distribution');

const svg = container.append('svg')

.attr('width', this.width + this.margin.left + this.margin.right)

.attr('height', this.height + this.margin.top + this.margin.bottom)

.append('g')

.attr('transform', `translate(${this.margin.left},${this.margin.top})`);

*// Get KRI dimensions*

const dimensions = Object.keys(this.data.claims[0].kri\_scores);

*// Calculate outlier counts per KRI (95th percentile)*

const kriData = dimensions.map(dim => {

const values = this.data.claims.map(d => d.kri\_scores[dim]);

const cutoff = d3.quantile(values, 0.95);

return {

name: dim,

outliers: values.filter(v => v >= cutoff).length,

total: values.length,

rate: values.filter(v => v >= cutoff).length / values.length

};

});

*// Create scales*

const x = d3.scaleBand()

.domain(kriData.map(d => d.name))

.range([0, this.width])

.padding(0.2);

const y = d3.scaleLinear()

.domain([0, d3.max(kriData, d => d.rate)])

.range([this.height, 0]);

*// Add violin plot areas*

dimensions.forEach(dim => {

const values = this.data.claims.map(d => d.kri\_scores[dim]);

const kde = kernelDensityEstimator(kernelEpanechnikov(0.1), x.bandwidth() / 2);

const density = kde(values);

const area = d3.area()

.x(d => x(dim) + x.bandwidth() / 2 + d[1] \* 30)

.y0(d => y(d[0]))

.y1(() => y(0));

svg.append('path')

.datum(density)

.attr('fill', '#3498db')

.attr('opacity', 0.2)

.attr('d', area);

});

*// Add bars*

svg.selectAll('.bar')

.data(kriData)

.enter().append('rect')

.attr('class', 'path-transition')

.attr('x', d => x(d.name))

.attr('y', d => y(d.rate))

.attr('width', x.bandwidth())

.attr('height', d => this.height - y(d.rate))

.attr('fill', d => d.rate > 0.1 ? 'var(--danger)' : 'var(--primary)')

.attr('rx', 3)

.attr('ry', 3);

*// Add axes*

svg.append('g')

.attr('transform', `translate(0,${this.height})`)

.call(d3.axisBottom(x));

svg.append('g')

.call(d3.axisLeft(y).ticks(5).tickFormat(d3.format('.0%')));

*// Add title*

svg.append('text')

.attr('x', this.width / 2)

.attr('y', -10)

.attr('text-anchor', 'middle')

.style('font-size', '14px')

.style('font-weight', 'bold')

.text('Outlier Rate by KRI (95th Percentile)');

*// Helper functions for kernel density estimation*

function kernelDensityEstimator(kernel, bandwidth) {

return function(sample) {

return x => [x, d3.mean(sample, v => kernel((x - v) / bandwidth)) / bandwidth];

};

}

function kernelEpanechnikov(k) {

return function(u) {

return Math.abs(u /= k) <= 1 ? 0.75 \* (1 - u \* u) / k : 0;

};

}

}

showTooltip(event, d) {

const tooltip = d3.select('#tooltip');

tooltip.style('opacity', 1)

.html(`<strong>Claim ID:</strong> ${d.id}<br>

<strong>Outlier:</strong> ${d.outlier ? 'Yes' : 'No'}`)

.style('left', (event.pageX + 10) + 'px')

.style('top', (event.pageY - 28) + 'px');

}

updateData(data, riskCutoff) {

this.data = data;

this.riskCutoff = riskCutoff;

this.render();

}

unload() {

*// Clean up if needed*

}

}

Decision Comparison Dashboard (js/dashboards/decisionComparison.js)

javascript

export default class DecisionComparisonDashboard {

constructor(container, data, riskCutoff) {

this.container = container;

this.data = data;

this.riskCutoff = riskCutoff;

this.margin = {top: 40, right: 30, bottom: 80, left: 60};

this.width = 800 - this.margin.left - this.margin.right;

this.height = 500 - this.margin.top - this.margin.bottom;

}

render() {

this.container.innerHTML = `

<div class="dashboard">

<div class="panel">

<div class="panel-header">Decision Flow Analysis</div>

<div class="chart-container" id="sankey-diagram"></div>

</div>

<div class="panel">

<div class="panel-header">Decision Discrepancies</div>

<div class="chart-container" id="discrepancy-matrix"></div>

</div>

</div>

`;

this.renderSankeyDiagram();

this.renderDiscrepancyMatrix();

}

renderSankeyDiagram() {

const container = d3.select('#sankey-diagram');

const svg = container.append('svg')

.attr('width', this.width + this.margin.left + this.margin.right)

.attr('height', this.height + this.margin.top + this.margin.bottom)

.append('g')

.attr('transform', `translate(${this.margin.left},${this.margin.top})`);

*// Prepare nodes*

const nodes = [

...new Set([

...this.data.claims.map(d => `Model: ${d.model\_decision}`),

...this.data.claims.map(d => `Assessor: ${d.assessor\_decision}`),

...this.data.claims.map(d => `Final: ${d.final\_decision}`)

])

].map(name => ({ name }));

*// Prepare links*

const links = [];

*// Model to Assessor links*

const modelAssessor = d3.rollup(

this.data.claims,

v => v.length,

d => `Model: ${d.model\_decision}`,

d => `Assessor: ${d.assessor\_decision}`

);

modelAssessor.forEach((targets, source) => {

targets.forEach((value, target) => {

links.push({

source: nodes.findIndex(n => n.name === source),

target: nodes.findIndex(n => n.name === target),

value,

type: 'model-assessor'

});

});

});

*// Assessor to Final links*

const assessorFinal = d3.rollup(

this.data.claims,

v => v.length,

d => `Assessor: ${d.assessor\_decision}`,

d => `Final: ${d.final\_decision}`

);

assessorFinal.forEach((targets, source) => {

targets.forEach((value, target) => {

links.push({

source: nodes.findIndex(n => n.name === source),

target: nodes.findIndex(n => n.name === target),

value,

type: 'assessor-final'

});

});

});

*// Sankey layout*

const sankey = d3.sankey()

.nodeWidth(15)

.nodePadding(10)

.extent([[1, 1], [this.width - 1, this.height - 6]]);

const { nodes: sankeyNodes, links: sankeyLinks } = sankey({

nodes: nodes.map(d => ({ ...d })),

links: links.map(d => ({ ...d }))

});

*// Color scales*

const nodeColor = d3.scaleOrdinal()

.domain(['Model', 'Assessor', 'Final'])

.range(['#3498db', '#f39c12', '#2ecc71']);

const linkColor = d3.scaleOrdinal()

.domain(['model-assessor', 'assessor-final'])

.range(['#7fb3d5', '#f8c471']);

*// Draw links*

svg.append('g')

.selectAll('path')

.data(sankeyLinks)

.enter().append('path')

.attr('d', d3.sankeyLinkHorizontal())

.attr('stroke', d => linkColor(d.type))

.attr('stroke-width', d => Math.max(1, d.width))

.attr('stroke-opacity', 0.6)

.attr('fill', 'none');

*// Draw nodes*

const node = svg.append('g')

.selectAll('rect')

.data(sankeyNodes)

.enter().append('rect')

.attr('x', d => d.x0)

.attr('y', d => d.y0)

.attr('height', d => d.y1 - d.y0)

.attr('width', d => d.x1 - d.x0)

.attr('fill', d => nodeColor(d.name.split(': ')[0]))

.attr('stroke', '#000')

.attr('stroke-opacity', 0.2);

*// Add node labels*

svg.append('g')

.selectAll('text')

.data(sankeyNodes)

.enter().append('text')

.attr('x', d => d.x0 < this.width / 2 ? d.x1 + 6 : d.x0 - 6)

.attr('y', d => (d.y1 + d.y0) / 2)

.attr('dy', '0.35em')

.attr('text-anchor', d => d.x0 < this.width / 2 ? 'start' : 'end')

.text(d => d.name)

.style('font-size', '12px')

.style('fill', 'var(--dark)');

*// Add title*

svg.append('text')

.attr('x', this.width / 2)

.attr('y', -10)

.attr('text-anchor', 'middle')

.style('font-size', '14px')

.style('font-weight', 'bold')

.text('Claim Decision Flow');

}

renderDiscrepancyMatrix() {

const container = d3.select('#discrepancy-matrix');

const svg = container.append('svg')

.attr('width', this.width + this.margin.left + this.margin.right)

.attr('height', this.height + this.margin.top + this.margin.bottom)

.append('g')

.attr('transform', `translate(${this.margin.left},${this.margin.top})`);

*// Get unique decisions*

const decisions = ['Approved', 'Denied', 'Pending'];

*// Calculate discrepancy matrix*

const matrix = decisions.map(modelDecision => {

return decisions.map(assessorDecision => {

return this.data.claims.filter(claim =>

claim.model\_decision === modelDecision &&

claim.assessor\_decision === assessorDecision

).length;

});

});

*// Create scales*

const x = d3.scaleBand()

.domain(decisions)

.range([0, this.width])

.padding(0.05);

const y = d3.scaleBand()

.domain(decisions)

.range([0, this.height])

.padding(0.05);

const color = d3.scaleSequentialLog(d3.interpolateBlues)

.domain([1, d3.max(matrix.flat())]);

*// Create cells*

svg.selectAll('.row')

.data(matrix)

.enter().append('g')

.attr('class', 'row')

.selectAll('.cell')

.data((d, i) => d.map((value, j) => ({

model: decisions[i],

assessor: decisions[j],

value

})))

.enter().append('rect')

.attr('class', 'cell')

.attr('x', d => x(d.assessor))

.attr('y', d => y(d.model))

.attr('width', x.bandwidth())

.attr('height', y.bandwidth())

.attr('fill', d => d.value > 0 ? color(d.value) : '#eee')

.attr('stroke', '#fff')

.attr('rx', 3)

.attr('ry', 3)

.on('mouseover', this.showMatrixTooltip)

.on('mouseout', () => d3.select('#tooltip').style('opacity', 0));

*// Add value labels*

svg.selectAll('.value')

.data(matrix.flatMap((row, i) =>

row.map((value, j) => ({

model: decisions[i],

assessor: decisions[j],

value

}))

))

.enter().append('text')

.attr('x', d => x(d.assessor) + x.bandwidth() / 2)

.attr('y', d => y(d.model) + y.bandwidth() / 2)

.attr('dy', '0.35em')

.attr('text-anchor', 'middle')

.style('font-size', '10px')

.style('fill', d => d.value > d3.max(matrix.flat()) / 2 ? 'white' : 'var(--dark)')

.text(d => d.value || '');

*// Add axes*

svg.append('g')

.attr('transform', `translate(0,${this.height})`)

.call(d3.axisBottom(x))

.selectAll('text')

.style('text-anchor', 'middle');

svg.append('g')

.call(d3.axisLeft(y));

*// Add labels*

svg.append('text')

.attr('x', this.width / 2)

.attr('y', -15)

.attr('text-anchor', 'middle')

.style('font-size', '12px')

.text('Assessor Decision');

svg.append('text')

.attr('transform', 'rotate(-90)')

.attr('y', -40)

.attr('x', -this.height / 2)

.attr('text-anchor', 'middle')

.style('font-size', '12px')

.text('Model Decision');

*// Add title*

svg.append('text')

.attr('x', this.width / 2)

.attr('y', -30)

.attr('text-anchor', 'middle')

.style('font-size', '14px')

.style('font-weight', 'bold')

.text('Decision Agreement Matrix');

}

showMatrixTooltip(event, d) {

const tooltip = d3.select('#tooltip');

tooltip.style('opacity', 1)

.html(`

<strong>Model:</strong> ${d.model}<br>

<strong>Assessor:</strong> ${d.assessor}<br>

<strong>Count:</strong> ${d.value}

`)

.style('left', (event.pageX + 10) + 'px')

.style('top', (event.pageY - 28) + 'px');

}

updateData(data, riskCutoff) {

this.data = data;

this.riskCutoff = riskCutoff;

this.render();

}

unload() {

*// Clean up if needed*

}

}

5. Data Loading Utility (js/utils/dataLoader.js)

javascript

export async function loadFWAData() {

try {

const response = await fetch('data/fwa-data.json');

if (!response.ok) {

throw new Error('Network response was not ok');

}

return await response.json();

} catch (error) {

console.error('Error loading FWA data:', error);

throw error;

}

}

export function filterDataByTime(data, timePeriod) {

const now = new Date();

let cutoffDate;

switch(timePeriod) {

case '7d': cutoffDate = new Date(now.setDate(now.getDate() - 7)); break;

case '30d': cutoffDate = new Date(now.setDate(now.getDate() - 30)); break;

case '90d': cutoffDate = new Date(now.setDate(now.getDate() - 90)); break;

default: cutoffDate = new Date(0);

}

return {

claims: data.claims.filter(claim =>

new Date(claim.service\_date) >= cutoffDate

),

providers: data.providers

};

}

export function calculateRiskCutoff(claims, thresholdPercent) {

const riskScores = claims.map(d => d.risk\_score);

return d3.quantile(riskScores, thresholdPercent / 100);

}

6. Helper Functions (js/utils/helpers.js)

javascript

*// Format currency values*

export function formatCurrency(value) {

return new Intl.NumberFormat('en-US', {

style: 'currency',

currency: 'USD',

minimumFractionDigits: 0,

maximumFractionDigits: 0

}).format(value);

}

*// Generate a color scale based on risk score*

export function getRiskColorScale(domain = [0, 1]) {

return d3.scaleSequential(d3.interpolateRdYlGn)

.domain([domain[1], domain[0]]); *// Reverse for red = high risk*

}

*// Calculate hexbin layout for geographic points*

export function createHexbin(width, height, radius) {

return d3.hexbin()

.radius(radius)

.extent([[0, 0], [width, height]]);

}

*// Generate a unique ID*

export function generateId(prefix = 'id') {

return `${prefix}-${Math.random().toString(36).substr(2, 9)}`;

}

*// Debounce function for resize events*

export function debounce(func, wait = 100) {

let timeout;

return function(...args) {

clearTimeout(timeout);

timeout = setTimeout(() => {

func.apply(this, args);

}, wait);

};

}

7. Sample Data (data/fwa-data.json)

json

{

"claims": [

{

"claim\_id": "CLM20230001",

"member\_id": "MEM1001",

"provider\_id": "PROV2001",

"service\_date": "2023-01-15",

"service\_category": "Inpatient",

"procedure\_code": "DRG001",

"diagnosis\_code": "ICD10-A01",

"billed\_amount": 12500,

"allowed\_amount": 9500,

"paid\_amount": 9200,

"los\_days": 4,

"model\_decision": "Approved",

"assessor\_decision": "Approved",

"final\_decision": "Approved",

"fraud\_flag": false,

"risk\_score": 0.35,

"kri\_scores": {

"LOS Score": 0.4,

"Cost Anomaly": 0.3,

"Frequency": 0.2,

"Clinical Mismatch": 0.1

}

},

{

"claim\_id": "CLM20230002",

"member\_id": "MEM1002",

"provider\_id": "PROV2002",

"service\_date": "2023-01-18",

"service\_category": "Outpatient",

"procedure\_code": "CPT99214",

"diagnosis\_code": "ICD10-E11",

"billed\_amount": 350,

"allowed\_amount": 250,

"paid\_amount": 240,

"model\_decision": "Denied",

"assessor\_decision": "Approved",

"final\_decision": "Pending",

"fraud\_flag": true,

"risk\_score": 0.85,

"kri\_scores": {

"LOS Score": 0.1,

"Cost Anomaly": 0.9,

"Frequency": 0.8,

"Clinical Mismatch": 0.7

}

}

],

"providers": [

{

"id": "PROV2001",

"name": "City General Hospital",

"type": "hospital",

"specialty": "General",

"address": "123 Main St",

"city": "Metropolis",

"state": "NY",

"zip": "10001",

"lat": 40.7128,

"lng": -74.0060,

"risk\_score": 0.4,

"claim\_count": 1250,

"last\_audit": "2022-11-15"

},

{

"id": "PROV2002",

"name": "Downtown Clinic",

"type": "clinic",

"specialty": "Primary Care",

"address": "456 Oak Ave",

"city": "Metropolis",

"state": "NY",

"zip": "10002",

"lat": 40.7100,

"lng": -74.0080,

"risk\_score": 0.8,

"claim\_count": 850,

"last\_audit": "2023-01-05"

}

]

}

8. Implementation Notes

1. **Modular Architecture**:
   * Each dashboard is a self-contained module
   * Clean separation of concerns
   * Easy to add new dashboards
2. **Interactive Features**:
   * Dynamic filtering
   * Tooltips with detailed information
   * Responsive design
3. **Advanced Visualizations**:
   * Parallel coordinates for multi-dimensional KRI analysis
   * Sankey diagrams for decision flows
   * Heatmaps for discrepancy analysis
   * Kernel density estimation for distribution visualization
4. **Performance Considerations**:
   * Efficient data processing
   * Smooth transitions
   * Responsive design for different screen sizes

To complete the project, you would implement similar modules for the remaining dashboards (Provider Watchlist, Length of Stay, Savings Potential, etc.) following the same pattern. Each module would export a class with render(), updateData(), and unload() methods.

Would you like me to provide implementations for any of the other dashboard modules or focus on specific aspects of the visualization techniques?