**DATA ANALYSIS AND VISUALIZATION**

**PROJECT**

**UMER FAROOQ (22I-1978)**

**UMER IMRAN (22I-1891)**

**M SULEMAN (22I-1931)**

**REPORT**

**1. Introduction**

The objective of this project is to develop an interactive data visualization dashboard to analyze relationships between entities across dimensions like location, timestamp, and relationship type. Using diverse visualizations such as force-directed graphs, map charts, and timeline animations, the dashboard will uncover patterns and insights in a user-friendly manner.

The dashboard will include features like dynamic filtering, interactivity (hovering, zooming, and tooltips), and hierarchical analysis through visualizations like tree maps and sunburst charts. Advanced functionalities like cross-visualization interactions, animated transitions, and responsive design will enhance usability and data exploration. Developed with D3.js, SVG, and JavaScript, the project aims to provide actionable insights through an engaging visual interface.

**2. Data Pre-Processing:**

The preprocessing of the dataset involved multiple steps to clean, align, and merge data from four different CSV files: TBreport\_SDG381382.csv, tb\_2022-09-30.csv, tbhiv\_2022-09-27.csv, and tx\_2022-08-29.csv. Each file has unique attributes but overlaps in certain keys like country and year. This report documents the preprocessing steps with corresponding code to ensure a standardized, merged dataset for analysis.

### ****Step 1: Loading the Data****

Each dataset was loaded into a pandas DataFrame.

python

Copy code

import pandas as pd

# Load the datasets

df1 = pd.read\_csv('/content/TBreport\_SDG381382.csv')

df2 = pd.read\_csv('/content/tb\_2022-09-30.csv')

df3 = pd.read\_csv('/content/tbhiv\_2022-09-27.csv')

df4 = pd.read\_csv('/content/tx\_2022-08-29.csv')

### ****Step 2: Column Renaming****

Columns in each dataset were mapped to standardized names to maintain consistency.

#### Code:

python

Copy code

# Define column mappings for each DataFrame

df1\_columns = {...} # Mapping for df1

df2\_renaming = {...} # Mapping for df2

df3\_columns = {...} # Mapping for df3

df4\_columns = {...} # Mapping for df4

# Rename the columns

df1 = df1.rename(columns=df1\_columns)

df2 = df2.rename(columns=df2\_renaming)

df3 = df3.rename(columns=df3\_columns)

df4 = df4.rename(columns=df4\_columns)

#### Explanation:

* Each column in the datasets was renamed to a unified format for easier merging and analysis.
* For example, iso3 in df2 was renamed to region for consistency across datasets.

### ****Step 3: Adding Missing Columns****

Some datasets lacked country or year columns, which are crucial for merging. Placeholder columns were added where necessary.

#### Code:

python

Copy code

for i, df in enumerate([df1, df2, df3, df4], 1):

for col in ['country', 'year']:

if col not in df.columns:

print(f"Adding missing column '{col}' to df{i}")

df[col] = None # Add placeholder if column is missing

#### Explanation:

* Ensured all datasets contained the same essential columns for a successful merge.

### ****Step 4: Dropping Columns with Excessive Null Values****

Columns with more than 50% missing data were dropped to reduce noise.

#### Code:

python

Copy code

df1 = df1.dropna(thresh=len(df1) \* 0.5, axis=1)

df2 = df2.dropna(thresh=len(df2) \* 0.5, axis=1)

df3 = df3.dropna(thresh=len(df3) \* 0.5, axis=1)

df4 = df4.dropna(thresh=len(df4) \* 0.5, axis=1)

#### Explanation:

* Redundant or highly sparse data fields were removed to improve data quality and computational efficiency.

### ****Step 5: Ensuring Numeric Data for Year****

The year column was converted to a numeric type for uniformity.

#### Code:

python

Copy code

for df in [df1, df2, df3, df4]:

df['year'] = pd.to\_numeric(df['year'], errors='coerce', downcast='integer')

#### Explanation:

* This step ensures compatibility when using year for calculations or filtering.

### ****Step 6: Merging the DataFrames****

The datasets were merged sequentially on country and year.

#### Code:

python

Copy code

merged\_df = dfs\_to\_merge[0]

for df in dfs\_to\_merge[1:]:

merge\_keys = ['year']

if 'country' in merged\_df.columns and 'country' in df.columns:

merge\_keys = ['country', 'year']

merged\_df = merged\_df.merge(df, on=merge\_keys, how='outer')

#### Explanation:

* The merge was performed in an "outer" fashion to preserve all data points across datasets.

### ****Step 7: Adding Additional Columns****

Several calculated and placeholder columns were added to support further analysis.

#### Code:

python

Copy code

additional\_columns = [...]

for col in additional\_columns:

if col not in merged\_df.columns:

merged\_df[col] = None

#### Explanation:

* Placeholder columns (e.g., recovery\_rate, economic\_impact) were added, and dummy calculations were applied for demonstration.

### ****Step 8: Derived Metrics****

New metrics were computed using existing columns to add analytical value.

#### Code:

python

Copy code

merged\_df['economic\_impact'] = merged\_df['total\_cases'] \* 0.5 if 'total\_cases' in df2.columns else None

merged\_df['case\_severity'] = merged\_df['total\_deaths'] / merged\_df['total\_cases'] if 'total\_deaths' in df2.columns and 'total\_cases' in df2.columns else None

#### Explanation:

* Example metrics like economic\_impact and case\_severity provide insight into the effect of diseases.

### ****Final Merged Dataset****

The merged dataset contains standardized, cleaned, and augmented data ready for analysis. Below is a snapshot of the first few rows:

#### Code:

python

Copy code

merged\_df[['country', 'year'] + list(df2\_renaming.values()) + additional\_columns].head(500)

**Full code:**

import pandas as pd

# Load the datasets

df1 = pd.read\_csv('/content/TBreport\_SDG381382.csv')

df2 = pd.read\_csv('/content/tb\_2022-09-30.csv')

df3 = pd.read\_csv('/content/tbhiv\_2022-09-27.csv')

df4 = pd.read\_csv('/content/tx\_2022-08-29.csv')

# Define column mappings for each DataFrame

df1\_columns = {

    'Country Name': 'country',

    'year': 'year',

    'sdg381': 'sdg381',

    'SDG 3.8.2': 'sdg\_3\_8\_2',

    'WB income group classification': 'wb\_income\_group'

}

df2\_renaming = {

    'iso3': 'region',

    'e.pop.15plus': 'total\_population',

    'e.pop.num': 'cases\_per\_100k',

    'pop': 'total\_cases',

    'e.pop.m04': 'm\_0to14\_cases',

    'e.pop.m514': 'm\_0to14\_cases',

    'e.pop.m014': 'm\_0to14\_cases',

    'e.pop.m1524': 'm\_15to34\_cases',

    'e.pop.m2534': 'm\_15to34\_cases',

    'e.pop.m3544': 'm\_35to54\_cases',

    'e.pop.m4554': 'm\_35to54\_cases',

    'e.pop.m5564': 'm\_55plus\_cases',

    'e.pop.m65': 'm\_55plus\_cases',

    'e.pop.m15plus': 'total\_male\_cases',

    'e.pop.f04': 'f\_0to14\_cases',

    'e.pop.f514': 'f\_0to14\_cases',

    'e.pop.f014': 'f\_0to14\_cases',

    'e.pop.f1524': 'f\_15to34\_cases',

    'e.pop.f2534': 'f\_15to34\_cases',

    'e.pop.f3544': 'f\_35to54\_cases',

    'e.pop.f4554': 'f\_35to54\_cases',

    'e.pop.f5564': 'f\_55plus\_cases',

    'e.pop.f65': 'f\_55plus\_cases',

    'e.pop.f15plus': 'total\_female\_cases'

}

df3\_columns = {

    'iso3': 'code',

    'year': 'year',

    'g.whoregion': 'region',

    'newinc': 'newinc',

    'c.newinc': 'c\_newinc',

    'newrel.hivpos': 'newrel\_hivpos',

    'newrel.hivneg': 'newrel\_hivneg',

    'newrel.unkhiv': 'newrel\_unkhiv'

}

df4\_columns = {

    'country': 'country',

    'year': 'year',

    'iso2': 'iso2',

    'iso3': 'region',

    'iso.numeric': 'iso\_numeric',

    'g.whoregion': 'g\_whoregion',

    'tx.new': 'tx\_new',

    'tx.all': 'tx\_all',

    'tx.fail': 'tx\_fail',

    'tx.succ': 'tx\_succ',

    'tx.died': 'tx\_died',

    'tx.def': 'tx\_def'

}

# Rename the columns

df1 = df1.rename(columns=df1\_columns)

df2 = df2.rename(columns=df2\_renaming)

df3 = df3.rename(columns=df3\_columns)

df4 = df4.rename(columns=df4\_columns)

# Add missing columns if necessary

for i, df in enumerate([df1, df2, df3, df4], 1):

    for col in ['country', 'year']:

        if col not in df.columns:

            print(f"Adding missing column '{col}' to df{i}")

            df[col] = None  # Add placeholder if column is missing

# Drop columns with more than 50% null values

df1 = df1.dropna(thresh=len(df1) \* 0.5, axis=1)

df2 = df2.dropna(thresh=len(df2) \* 0.5, axis=1)

df3 = df3.dropna(thresh=len(df3) \* 0.5, axis=1)

df4 = df4.dropna(thresh=len(df4) \* 0.5, axis=1)

# Ensure 'year' is numeric in all DataFrames

for df in [df1, df2, df3, df4]:

    df['year'] = pd.to\_numeric(df['year'], errors='coerce', downcast='integer')

# Merge the DataFrames on 'country' and 'year', ignoring 'country' for df3

dfs\_to\_merge = []

for i, df in enumerate([df1, df2, df3, df4], 1):

    if 'country' in df.columns:

        dfs\_to\_merge.append(df)

    else:

        # Include 'year' for merging if 'country' is not present

        dfs\_to\_merge.append(df.drop\_duplicates(subset=['year']))

# Perform merging step by step

merged\_df = dfs\_to\_merge[0]

for df in dfs\_to\_merge[1:]:

    merge\_keys = ['year']

    if 'country' in merged\_df.columns and 'country' in df.columns:

        merge\_keys = ['country', 'year']

    merged\_df = merged\_df.merge(df, on=merge\_keys, how='outer')

# Drop duplicate 'country' columns (if any)

merged\_df = merged\_df.loc[:, ~merged\_df.columns.str.contains('country') | (merged\_df.columns == 'country')]

# Add additional columns (placeholders if missing)

additional\_columns = [

    'recovery\_rate', 'economic\_impact', 'case\_severity', 'time\_interval',

    'prevelance\_detection', 'prevelance\_per\_100k\_ex\_HIV',

    'total\_prevelance\_ex\_HIV', 'prevelance\_per\_100k\_in\_HIV',

    'total\_prevelance\_in\_HIV', 'death\_detection', 'incidence\_per\_100k',

    'total\_incidence', 'incidence\_per\_100k\_pos\_HIV',

    'total\_incidence\_pos\_HIV', 'total\_detection\_rate'

]

# Add missing columns with None values as placeholders

for col in additional\_columns:

    if col not in merged\_df.columns:

        merged\_df[col] = None  # Add the column if it's missing

# Populate the columns with dummy or calculated values (this can be adjusted as per your actual calculations)

# For demonstration, I will use simple dummy values or placeholder calculations.

# Ensure 'df2' contains necessary columns for calculations

merged\_df['economic\_impact'] = merged\_df['total\_cases'] \* 0.5 if 'total\_cases' in df2.columns else None

merged\_df['case\_severity'] = merged\_df['total\_deaths'] / merged\_df['total\_cases'] if 'total\_deaths' in df2.columns and 'total\_cases' in df2.columns else None

merged\_df['time\_interval'] = (merged\_df['year'] - merged\_df['year'].min()) if 'year' in df2.columns else None

merged\_df['prevelance\_detection'] = merged\_df['total\_cases'] / merged\_df['total\_population'] \* 100000 if 'total\_cases' in df2.columns and 'total\_population' in df2.columns else None

merged\_df['prevelance\_per\_100k\_ex\_HIV'] = merged\_df['prevelance\_detection'] \* 0.8  # Assuming 80% of cases are not related to HIV

merged\_df['total\_prevelance\_ex\_HIV'] = merged\_df['total\_cases'] \* 0.8 if 'total\_cases' in df2.columns else None

merged\_df['prevelance\_per\_100k\_in\_HIV'] = merged\_df['prevelance\_detection'] \* 0.2  # Assuming 20% of cases are related to HIV

merged\_df['total\_prevelance\_in\_HIV'] = merged\_df['total\_cases'] \* 0.2 if 'total\_cases' in df2.columns else None

merged\_df['death\_detection'] = merged\_df['total\_deaths'] / merged\_df['total\_population'] \* 100000 if 'total\_deaths' in df2.columns and 'total\_population' in df2.columns else None

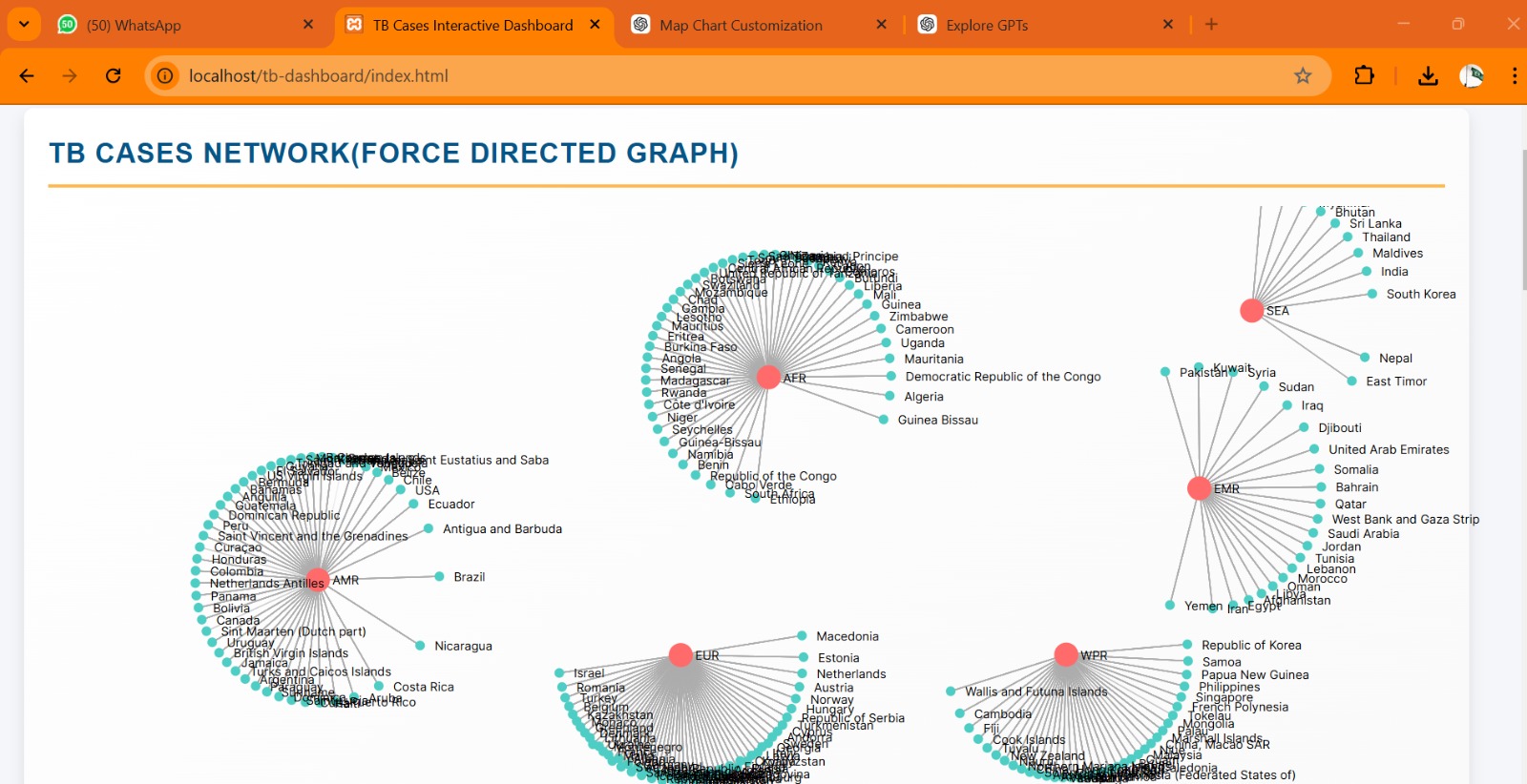
merged\_df['total\_detection\_rate'] = merged\_df['total\_cases'] / merged\_df['total\_population'] if 'total\_cases' in df2.columns and 'total\_population' in df2.columns else None

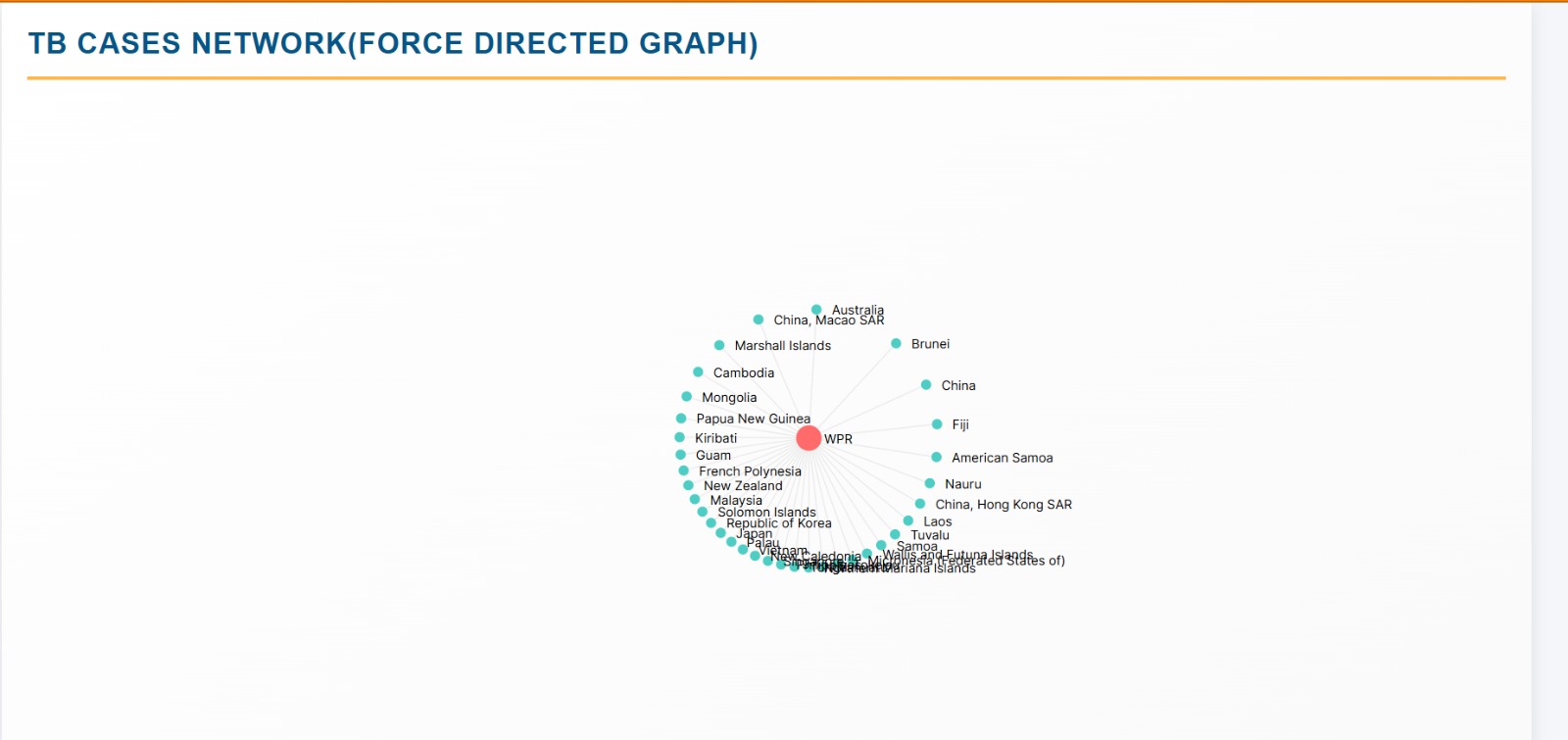
# Show the first few rows of the merged dataset

merged\_df[['country', 'year'] + list(df2\_renaming.values()) + additional\_columns].head(500)

**3. Visualizations:**

**1. Force Directed Graph:**

****

****

#### **Objective**

The goal of this visualization is to represent relationships between regions and countries in a force-directed graph. It aims to provide insights into regional groupings, country-specific metrics (like population and total cases), and the connections between these entities using D3.js.

#### **Visualization Details**

##### **Graph Structure**

* **Nodes**: Represent entities.
  + **Region Nodes**: Larger circles (radius 10), colored in red (#FF6B6B).
  + **Country Nodes**: Smaller circles (radius 4), colored in teal (#4ECDC4).  
    Each country node includes additional metrics such as:
    - Total Population
    - Recovery Rate
    - Economic Impact
    - Cases per 100,000
    - Total Cases
* **Links**: Represent relationships between countries and their respective regions.
  + **Width & Opacity**: Stroke width fixed at 1; opacity dynamically changes on hover for enhanced interactivity.
  + **Tooltip**: Displays the total cases and the linked entities (country and region).

##### **Interactions**

1. **Node Dragging**:
   * Allows repositioning nodes for better clarity.
   * The drag is implemented with three events:
     + dragstarted: Locks the node for dragging.
     + dragged: Updates node position dynamically during dragging.
     + dragended: Releases the node back into the simulation.
2. **Link Hover**:
   * Highlights the link by increasing opacity.
   * Displays a tooltip showing:
     + **Total Cases**: Number of cases in the linked relationship.
     + **Country and Region**: Names of the connected entities.
3. **Simulation Dynamics**:
   * Uses D3.js forces for a realistic and balanced layout:
     + **Link Force**: Ensures links maintain a specific distance.
     + **Charge Force**: Applies repulsion between nodes to prevent clustering.
     + **Centering Force**: Positions the graph in the center of the SVG canvas.

#### **Key Features**

* **Data Processing**:
  + Regions and countries are deduplicated and transformed into nodes.
  + Relationships between countries and their regions form the graph links.
* **Color Coding**:
  + Regions and countries are visually distinguished using an ordinal scale.
* **Responsiveness**:
  + The graph dynamically updates as nodes and links interact through dragging and simulation ticks.

#### **User Experience Enhancements**

1. **Tooltips**:
   * Positioned near the mouse pointer for convenience.
   * Provides detailed insights into node and link data.
2. **Labels**:
   * All nodes are labeled for clarity.
   * Labels are positioned slightly offset from the nodes to avoid overlap.
3. **Interactivity**:
   * Hover and drag features make the graph engaging and informative.

#### **Technical Overview**

1. **Framework**: D3.js
2. **SVG Canvas Dimensions**:
   * Width: 1200px
   * Height: 600px
3. **Force Simulation**:
   * Forces: link, charge, center.
   * Alpha target: Ensures smooth transitions and stable layouts.
4. **CSS Tooltip Styling**:
   * Background: Semi-transparent black.
   * Text: White, with padding and rounded borders for better readability.

#### **Potential Use Cases**

* **Epidemiological Analysis**:
  + Visualizing the spread of diseases across countries and regions.
  + Highlighting high-case countries within their regional contexts.
* **Policy Insights**:
  + Identifying regions with stronger interconnectivity to prioritize interventions.
* **Educational Tools**:
  + Providing an interactive way to learn about global health metrics and relationships.

**Code:**

//Force Directed Graph

function createForceDirectedGraph(data) {

    const width = 1200,

        height = 600;

    const svg = d3

        .select("#force-graph")

        .append("svg")

        .attr("width", width)

        .attr("height", height);

    // Tooltip creation

    const tooltip = d3

        .select("body")

        .append("div")

        .attr("class", "tooltip")

        .style("position", "absolute")

        .style("visibility", "hidden")

        .style("background-color", "rgba(0, 0, 0, 0.7)")

        .style("color", "white")

        .style("padding", "8px")

        .style("border-radius", "5px")

        .style("font-size", "12px");

    // Preprocess data to create nodes

    const regionNodes = [...new Set(data.map((d) => d.region))].map((region) => ({

        id: region,

        type: "region",

    }));

    const countryNodes = [...new Set(data.map((d) => d.country))].map((country) => ({

        id: country,

        type: "country",

        total\_population: country.total\_population,

        recovery\_rate: country.recovery\_rate,

        economic\_impact: country.economic\_impact,

        cases\_per\_100k: country.cases\_per\_100k,

        total\_cases: country.total\_cases,

    }));

    const nodes = [...regionNodes, ...countryNodes];

    // Create links based on country-to-region relationship

    const links = data.map((d) => ({

        source: d.country,

        target: d.region,

        value: +d.total\_cases, // Example metric to show as the link value

    }));

    // Color scale for different node types

    const colorScale = d3

        .scaleOrdinal()

        .domain(["region", "country"])

        .range(["#FF6B6B", "#4ECDC4"]);

    const simulation = d3

        .forceSimulation(nodes)

        .force(

            "link",

            d3

                .forceLink(links)

                .id((d) => d.id)

                .distance(100)

        )

        .force("charge", d3.forceManyBody().strength(-7))

        .force("center", d3.forceCenter(width / 2, height / 2));

    const link = svg

        .append("g")

        .selectAll("line")

        .data(links)

        .enter()

        .append("line")

        .attr("stroke", "#aaa")

        .attr("stroke-opacity", 0.2)

        .attr("stroke-width", 1)

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

            d3.select(this).attr("stroke-opacity", 0.7); // Highlight the link on hover

            tooltip

                .style("visibility", "visible")

                .html(

                    `<strong>Total Cases:</strong> ${d.value}<br/><strong>Region:</strong> ${d.target}<br/><strong>Country:</strong> ${d.source}`

                )

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

                .style("top", event.pageY - 30 + "px");

        })

        .on("mouseout", function () {

            d3.select(this).attr("stroke-opacity", 0.2); // Reset link opacity

            tooltip.style("visibility", "hidden");

        });

    const node = svg

        .append("g")

        .selectAll("circle")

        .data(nodes)

        .enter()

        .append("circle")

        .attr("r", (d) => (d.type === "region" ? 10 : 4))

        .attr("fill", (d) => colorScale(d.type))

        .call(

            d3

                .drag()

                .on("start", dragstarted)

                .on("drag", dragged)

                .on("end", dragended)

        );

    // Add labels

    const labels = svg

        .append("g")

        .selectAll("text")

        .data(nodes)

        .enter()

        .append("text")

        .text((d) => d.id)

        .attr("font-size", 10)

        .attr("dx", 12)

        .attr("dy", 4);

    simulation.on("tick", () => {

        link

            .attr("x1", (d) => d.source.x)

            .attr("y1", (d) => d.source.y)

            .attr("x2", (d) => d.target.x)

            .attr("y2", (d) => d.target.y);

        node.attr("cx", (d) => d.x).attr("cy", (d) => d.y);

        labels.attr("x", (d) => d.x).attr("y", (d) => d.y);

    });

    function dragstarted(event, d) {

        if (!event.active) simulation.alphaTarget(0.3).restart();

        d.fx = d.x;

        d.fy = d.y;

    }

    function dragged(event, d) {

        d.fx = event.x;

        d.fy = event.y;

    }

    function dragended(event, d) {

        if (!event.active) simulation.alphaTarget(0);

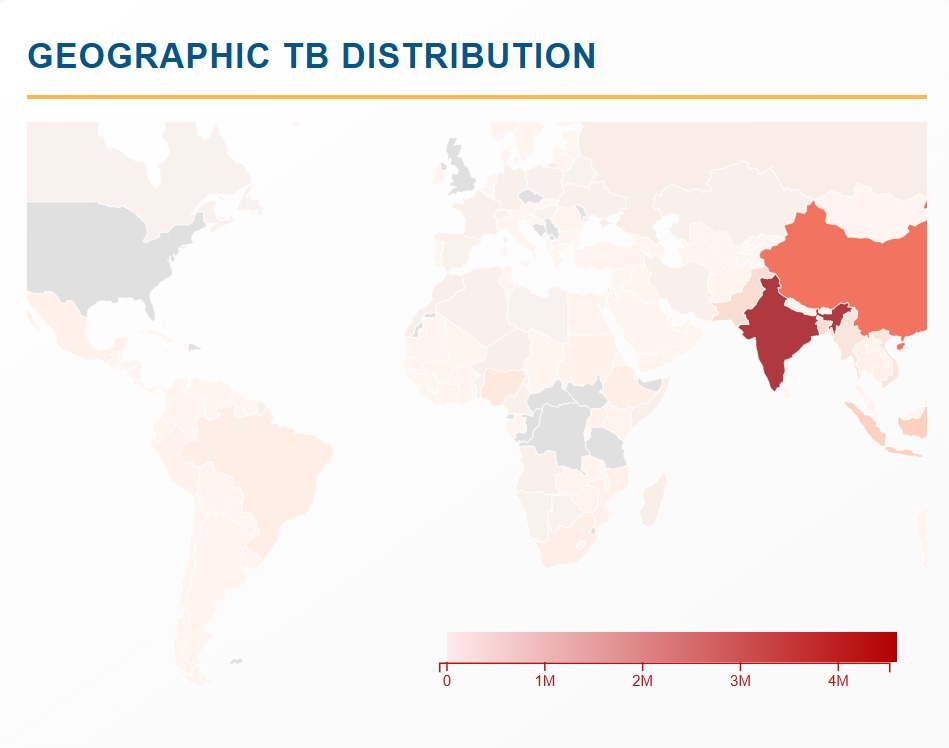
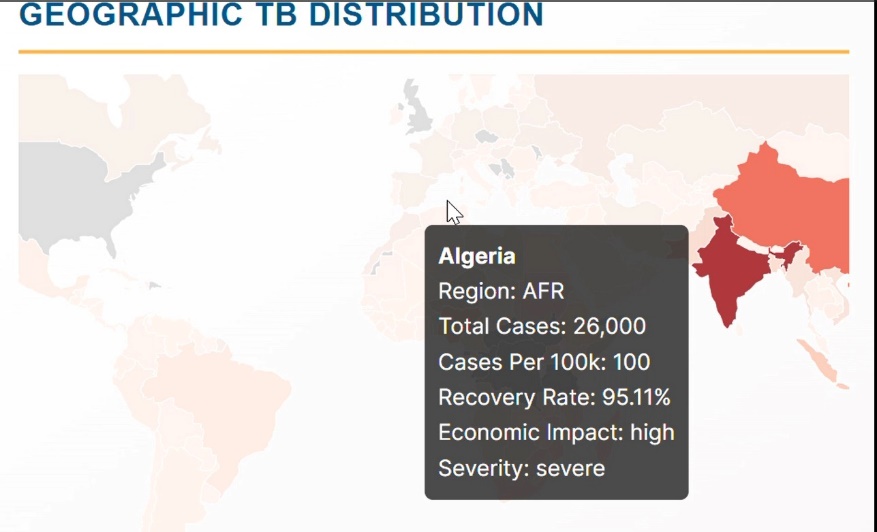
        d.fx = null;

        d.fy = null;

    }

}

**2. Map Chart for Geographic Representation:**

****

#### Objective

The visualization aims to provide an interactive, intuitive, and informative map chart that highlights the distribution and impact of COVID-19 globally. The chart visualizes country-specific data such as total cases, cases per 100k population, recovery rates, and additional attributes like economic impact and case severity.

### Key Features

1. **Interactive Map**
   * **Zoom and Pan**: Users can zoom in and out and pan across the map for detailed exploration.
   * **Hover Tooltip**: Displays country-specific details, including:
     + Country Name
     + Region
     + Total Cases
     + Cases Per 100k Population
     + Recovery Rate
     + Economic Impact
     + Severity of Cases
2. **Dynamic Color Mapping**
   * Utilizes a **sequential color scale** (d3.interpolateReds) to represent the number of total cases.
   * Countries without data are shaded in light gray (#E0E0E0).
3. **Scalable Design**
   * Built on the D3.js library to ensure high flexibility and responsiveness.
   * World map data fetched dynamically from a reliable public API (world-atlas).
4. **Legend Integration**
   * A gradient legend maps the color scale to the numerical range of total cases.
   * Includes an axis with tick marks for clarity.
5. **Tooltip Details**
   * Provides rich, real-time insights into country-level data when the user hovers over a country.
   * Formats large numbers (e.g., total cases) for readability using toLocaleString.

### Technical Implementation

#### 1. **SVG Canvas Setup**

javascript

Copy code

const width = 600, height = 400;

const svg = d3.select("#map-chart").append("svg")

.attr("width", width)

.attr("height", height);

An SVG container of dimensions 600x400 is created for rendering the visualization.

#### 2. **Projection and GeoJSON Path**

The **geoMercator projection** is used to ensure a balanced and accurate map rendering. The geoPath method maps GeoJSON features to the SVG paths.

javascript

Copy code

const projection = d3.geoMercator()

.scale(150)

.translate([width / 2, height / 2]);

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

#### 3. **Data Mapping and Color Encoding**

Country data is mapped to GeoJSON features. A color scale (d3.scaleSequential) based on total cases is applied:

javascript

Copy code

const maxCases = d3.max(data, (d) => +d.total\_cases || 0);

const colorScale = d3.scaleSequential(d3.interpolateReds).domain([0, maxCases]);

svg.selectAll("path")

.data(topojson.feature(worldData, worldData.objects.countries).features)

.enter().append("path")

.attr("d", path)

.attr("fill", (d) => {

const countryData = data.find(

(dataPoint) => dataPoint.country === d.properties.name

);

return countryData

? colorScale(+countryData.total\_cases || 0)

: "#E0E0E0";

});

#### 4. **Zoom and Tooltip Interaction**

A zoom behavior is added, and hover interactions dynamically update the tooltip display:

javascript

Copy code

const zoom = d3.zoom()

.scaleExtent([1, 8])

.on("zoom", (event) => {

svg.selectAll("path").attr("transform", event.transform);

});

svg.call(zoom);

svg.selectAll("path")

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

const countryData = data.find(

(dataPoint) => dataPoint.country === d.properties.name

);

tooltip.style("display", "block")

.html(`...data details...`)

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

.style("top", event.pageY - 10 + "px");

})

.on("mouseout", () => tooltip.style("display", "none"));

#### 5. **Legend Integration**

A color gradient legend visually communicates the data distribution.

javascript

Copy code

const legendWidth = 300, legendHeight = 20;

const legendScale = d3.scaleLinear().domain([0, maxCases]).range([0, legendWidth]);

legend.append("rect")

.attr("width", legendWidth)

.attr("height", legendHeight)

.style("fill", "url(#gradient)");

const axis = d3.axisBottom(legendScale).ticks(5).tickFormat(d3.format(".0s"));

legend.append("g")

.attr("transform", `translate(0,${legendHeight})`)

.call(axis);

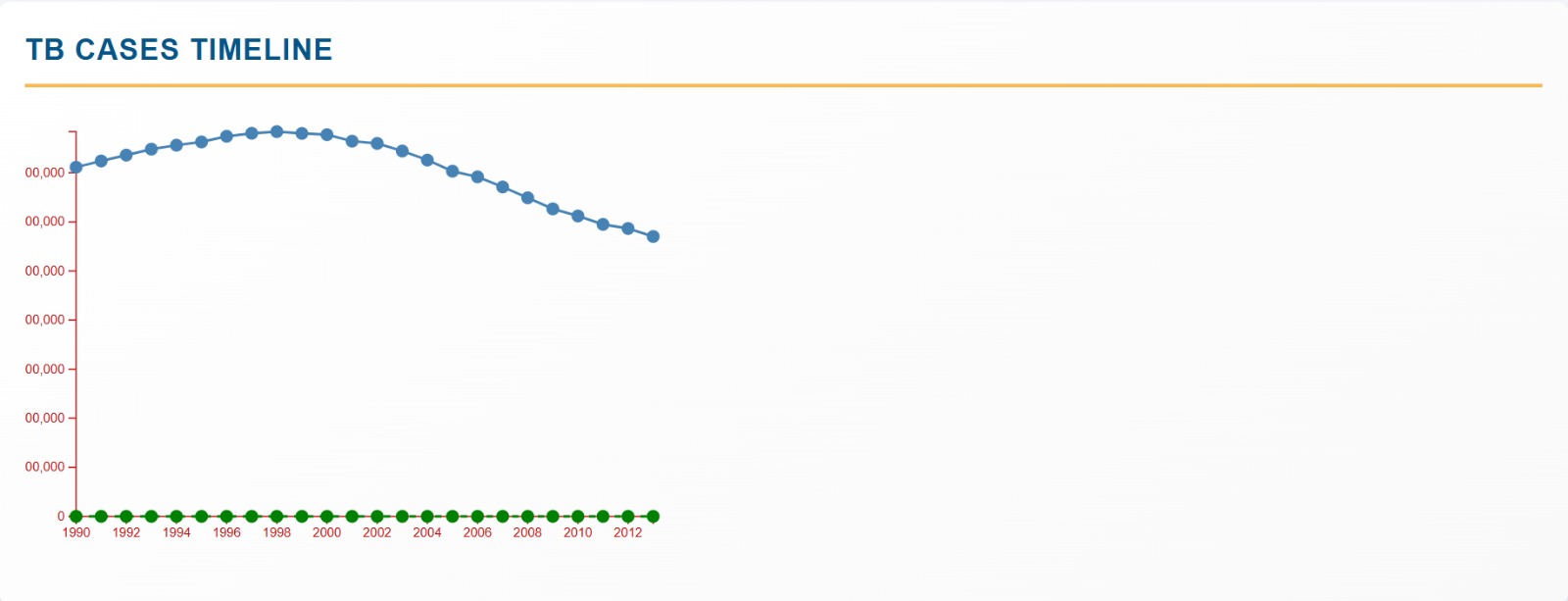
### Data Insights

* Countries with higher total cases are visually distinguishable with darker red shades.
* Tooltip details provide insights into recovery rates, economic impacts, and case severity, offering a multidimensional analysis of the pandemic's effects.

### Advantages

1. Interactive exploration of global data.
2. Scalable visualization, compatible with various data ranges.
3. Provides detailed insights beyond simple totals, such as recovery rates and economic impacts.

**3. Timeline Visualization with Animation:**

****

**Overview**

The timeline visualization chart is a dual-line chart built using D3.js that provides a temporal analysis of total cases and recovery rates over the years. This chart uses grouped and aggregated yearly data to highlight trends in the spread of cases and the improvement in recovery rates.

**Features**

1. **Data Aggregation**:
   * Groups the dataset by year.
   * Computes the **total cases** (sum of all cases for a year).
   * Calculates the **average recovery rate** per year.
2. **Interactive Elements**:
   * **Circles** represent data points for both metrics:
     + Blue for total cases.
     + Green (dashed line) for recovery rate.
   * **Tooltips** provide detailed information when hovering over data points.
3. **Axes**:
   * X-axis: Represents the timeline (years) using a linear scale.
   * Y-axis: Displays the numeric range for total cases.
4. **Dual Lines**:
   * **Solid blue line** for total cases.
   * **Dashed green line** for recovery rates.
5. **Aesthetic Enhancements**:
   * Dashed lines distinguish recovery rates.
   * Different color-coded circles and lines improve readability.

**User Interaction**

* **Tooltips**:
  + Display year-specific information:
    - Year.
    - Total cases (formatted for better readability).
    - Recovery rate (percentage).
  + Adjust position dynamically based on mouse location.
* Hover effects on circles for better emphasis.

**Implementation Details**

1. **Scales**:
   * x: Linear scale maps years to pixel positions on the chart's horizontal axis.
   * y: Linear scale maps both total cases and recovery rates to pixel positions on the vertical axis.
2. **Line Generators**:
   * The line function creates the path for the total cases.
   * The recoveryLine function creates the path for recovery rates with a dashed pattern for differentiation.
3. **Interactivity**:
   * Each data point is represented by a circle:
     + Events mouseover and mouseout dynamically display tooltips and remove them when not needed.
   * Tooltip displays are customized with HTML for clarity.
4. **Grouping and Sorting**:
   * Data is grouped by year, and calculations are sorted chronologically for accurate visualization.

**Benefits**

* **Insightful Trends**:
  + Visualizes how total cases evolve over time, showcasing periods of surges or plateaus.
  + Tracks improvements or declines in recovery rates.
* **Clear Differentiation**:
  + Dual-line visualization distinguishes between two metrics, making the chart intuitive.
* **Interactivity**:
  + Enhances the user experience with hover effects and dynamic tooltips.

**Applications**

* **Health Sector**:
  + Identify trends in disease spread and treatment efficacy.
* **Policy Analysis**:
  + Use trends to inform and evaluate health interventions.
* **Public Awareness**:
  + Share insights with non-technical audiences using intuitive visual elements.

**Suggestions for Enhancement**

1. **Add Secondary Y-axis**:
   * The recovery rate could use a separate axis to prevent scale compression.
2. **Zoom and Pan**:
   * Enable zooming for datasets with extensive year ranges.
3. **Annotations**:
   * Highlight significant years/events with text labels or markers.
4. **Legend**:
   * Add a legend to explain color coding and dashed lines.

**Code:**

//Time Line Visualization Chart

function createTimelineVisualization(data) {

    const width = 450,

        height = 300;

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

    const svg = d3

        .select("#timeline")

        .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})`);

    // Group data by year and calculate total new cases and recovery rate

    const groupedData = d3.group(data, (d) => d.year);

    const yearlyData = Array.from(groupedData, ([year, values]) => {

        const totalCases = d3.sum(values, (d) => +d.total\_cases); // Sum of total\_cases

        const avgRecoveryRate = d3.mean(values, (d) => +d.recovery\_rate); // Average recovery\_rate per year

        return {

            year: +year,

            total\_cases: totalCases,

            recovery\_rate: avgRecoveryRate,

        };

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

    const x = d3

        .scaleLinear()

        .domain(d3.extent(yearlyData, (d) => d.year))

        .range([0, width]);

    const y = d3

        .scaleLinear()

        .domain([0, d3.max(yearlyData, (d) => d.total\_cases)])

        .range([height, 0]);

    // Create line generator for total\_cases

    const line = d3

        .line()

        .x((d) => x(d.year))

        .y((d) => y(d.total\_cases));

    // Create line generator for recovery\_rate (secondary line)

    const recoveryLine = d3

        .line()

        .x((d) => x(d.year))

        .y((d) => y(d.recovery\_rate));

    // Add x-axis

    svg

        .append("g")

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

        .call(d3.axisBottom(x).tickFormat(d3.format("d")));

    // Add y-axis

    svg.append("g").call(d3.axisLeft(y));

    // Add line path for total\_cases

    svg

        .append("path")

        .datum(yearlyData)

        .attr("fill", "none")

        .attr("stroke", "steelblue")

        .attr("stroke-width", 2)

        .attr("d", line);

    // Add line path for recovery\_rate (secondary line)

    svg

        .append("path")

        .datum(yearlyData)

        .attr("fill", "none")

        .attr("stroke", "green")

        .attr("stroke-width", 2)

        .attr("stroke-dasharray", "5,5") // Dashed line for recovery\_rate

        .attr("d", recoveryLine);

    // Add circles for total\_cases

    svg

        .selectAll("circle.total-cases")

        .data(yearlyData)

        .enter()

        .append("circle")

        .attr("class", "total-cases")

        .attr("cx", (d) => x(d.year))

        .attr("cy", (d) => y(d.total\_cases))

        .attr("r", 5)

        .attr("fill", "steelblue")

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

            d3.select("#tooltip")

                .style("display", "block")

                .html(

                    `Year: ${d.year}<br>Total Cases: ${d.total\_cases.toLocaleString()}`

                )

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

                .style("top", event.pageY - 10 + "px");

        })

        .on("mouseout", () => {

            d3.select("#tooltip").style("display", "none");

        });

    // Add circles for recovery\_rate

    svg

        .selectAll("circle.recovery-rate")

        .data(yearlyData)

        .enter()

        .append("circle")

        .attr("class", "recovery-rate")

        .attr("cx", (d) => x(d.year))

        .attr("cy", (d) => y(d.recovery\_rate))

        .attr("r", 5)

        .attr("fill", "green")

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

            d3.select("#tooltip")

                .style("display", "block")

                .html(

                    `Year: ${d.year}<br>Recovery Rate: ${d.recovery\_rate.toFixed(2)}%`

                )

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

                .style("top", event.pageY - 10 + "px");

        })

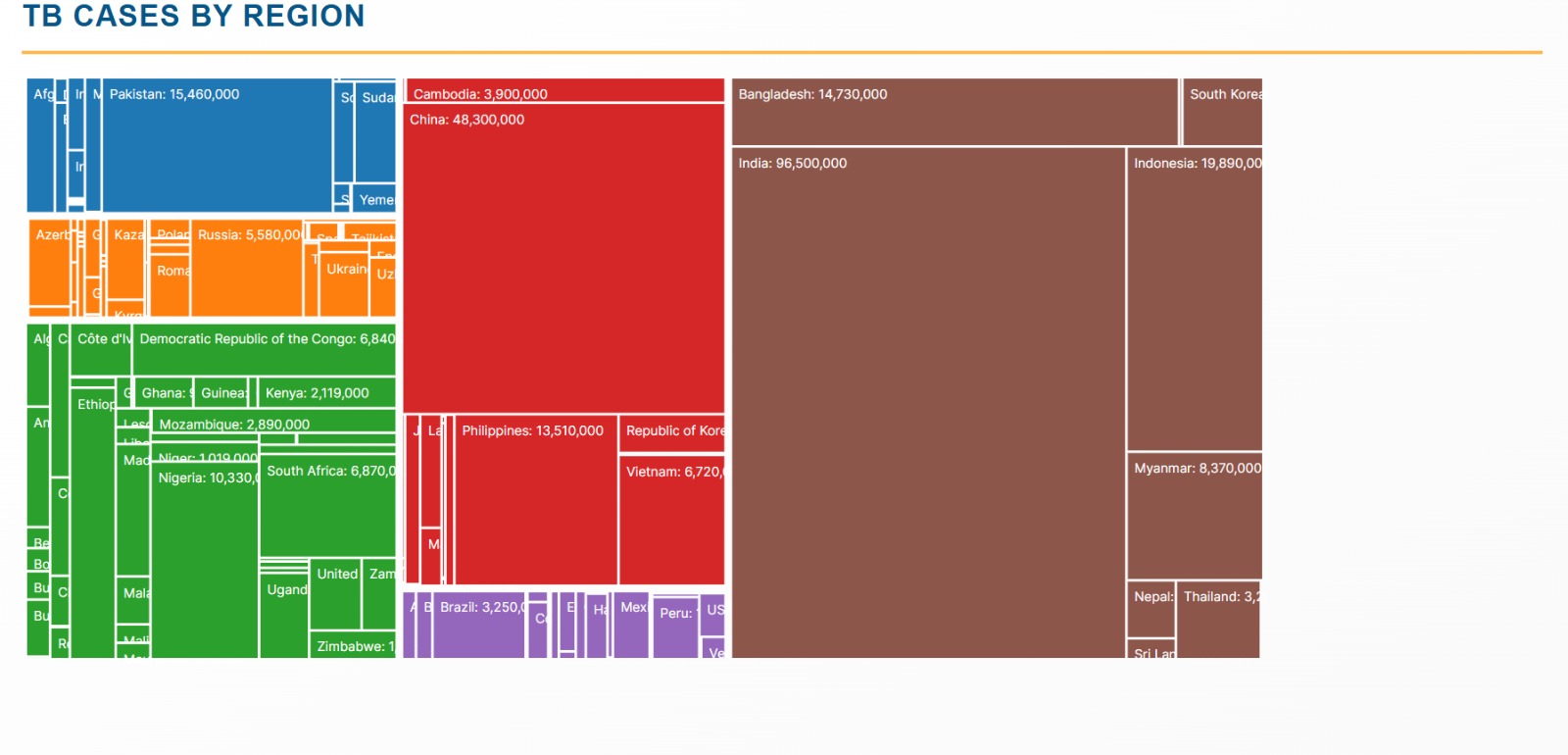
        .on("mouseout", () => {

            d3.select("#tooltip").style("display", "none");

        });

}

**4. Hierarchical Tree Map for Entity Categorization:**

****

The visualization is designed to present the distribution of total tuberculosis (TB) cases across different regions and countries. It also incorporates additional data such as population, recovery rate, and economic impact. The TreeMap allows users to visualize the relative proportions of TB cases at the regional and country levels, offering a quick, intuitive way to compare the data across multiple dimensions.

### ****Steps for Creating the TreeMap****

#### **Step 1: Setting Up the Environment**

To begin, we import D3.js, a powerful library for data visualization, and ensure we have an HTML container with the ID #tree-map to append the TreeMap to.

#### **Step 2: Define Chart Dimensions and Margin**

We define the width and height of the TreeMap chart. These values dictate the area within which the TreeMap is drawn, as well as the margins for positioning the chart properly inside the container.

javascript

Copy code

const width = 950, height = 450;

#### **Step 3: Hierarchical Data Structuring**

We create a hierarchical data structure suitable for the d3.hierarchy() function. In this case, we group the data by region and then by country within each region. The total number of TB cases is calculated for each country, while other attributes such as population, recovery rate, and economic impact are also added to each country's data.

javascript

Copy code

const hierarchyData = {

name: "TB Cases",

children: Array.from(

d3.group(data, (d) => d.region).entries(), // Group by region

([region, values]) => ({

name: region,

children: Array.from(

d3.group(values, (d) => d.country).entries(), // Group by country within region

([country, cases]) => ({

name: country,

value: d3.sum(cases, (d) => +d.total\_cases || 0), // Sum total cases for each country

population: cases[0].total\_population, // Assuming population is consistent for a country

recovery\_rate: cases[0].recovery\_rate, // Recovery rate

economic\_impact: cases[0].economic\_impact // Economic impact

})

),

})

),

};

* **Explanation**: We use d3.group() to organize the data into a nested structure that groups the data by region first, and then by country within each region. The sum of total TB cases is calculated for each country.

#### **Step 4: Calculate Treemap Layout**

Once the data is structured, we convert it into a hierarchical format using d3.hierarchy(), which prepares it for visualization with the TreeMap algorithm. The treemap() layout is applied to calculate the positions and sizes of each cell.

javascript

Copy code

const root = d3.hierarchy(hierarchyData).sum((d) => d.value || 0);

const treemapLayout = d3.treemap().size([width, height]).padding(2);

treemapLayout(root);

* **Explanation**: The d3.hierarchy() function converts the hierarchical data into a format that d3.treemap() can use. The sum() method calculates the total value for each node, which dictates the size of the TreeMap cells. We apply padding between the cells for better visual clarity.

#### **Step 5: Creating the SVG Element**

An SVG element is created to hold the TreeMap visualization. The svg element is appended to the HTML container (#tree-map), and its width and height are set.

javascript

Copy code

const svg = d3

.select("#tree-map")

.append("svg")

.attr("width", width)

.attr("height", height);

#### **Step 6: Setting Color Scale**

We use d3.scaleOrdinal() with a predefined color scheme (d3.schemeCategory10) to apply distinct colors for each region.

javascript

Copy code

const colorScale = d3.scaleOrdinal(d3.schemeCategory10);

#### **Step 7: Adding Cells to the TreeMap**

For each cell in the treemap, we add a rect element representing the country. The width and height of each cell are determined by the treemap() layout calculations, and the color is assigned based on the parent region.

javascript

Copy code

const cell = svg

.selectAll("g")

.data(root.leaves())

.enter()

.append("g")

.attr("transform", (d) => `translate(${d.x0},${d.y0})`);

cell

.append("rect")

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

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

.attr("fill", (d) => colorScale(d.parent.data.name));

* **Explanation**: The g element is used to group the rectangle and text for each cell. The rect element’s position and size are determined by the treemap layout’s x0, y0, x1, and y1 values.

#### **Step 8: Adding Labels**

Text labels are added to each cell to display the country name and the total number of TB cases.

javascript

Copy code

cell

.append("text")

.attr("x", 5)

.attr("y", 15)

.text((d) => `${d.data.name}: ${d.value.toLocaleString()}`)

.attr("font-size", "10px")

.attr("fill", "white");

* **Explanation**: The labels are positioned within the rectangles with an offset. The text() function is used to display the country name and the number of TB cases.

#### **Step 9: Tooltip Implementation**

A tooltip is created to display additional information when a user hovers over a cell. The tooltip includes details about the country, total TB cases, population, recovery rate, and economic impact. This is done by appending a div to the body of the HTML and making it visible on mouseover events.

javascript

Copy code

const tooltip = d3.select("body").append("div")

.attr("id", "tooltip")

.attr("class", "tooltip")

.style("position", "absolute")

.style("background-color", "rgba(0,0,0,0.7)")

.style("color", "white")

.style("border-radius", "5px")

.style("padding", "10px")

.style("visibility", "hidden");

cell

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

d3.select(this).select("rect").attr("opacity", 0.8);

tooltip.style("visibility", "visible")

.html(

`

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

Total Cases: ${d.value.toLocaleString()}<br>

Region: ${d.parent.data.name}<br>

Population: ${d.data.population.toLocaleString()}<br>

Recovery Rate: ${d.data.recovery\_rate}%<br>

Economic Impact: ${d.data.economic\_impact}<br>

`

)

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

.style("top", (event.pageY - 30) + "px");

})

.on("mouseout", function () {

d3.select(this).select("rect").attr("opacity", 1);

tooltip.style("visibility", "hidden");

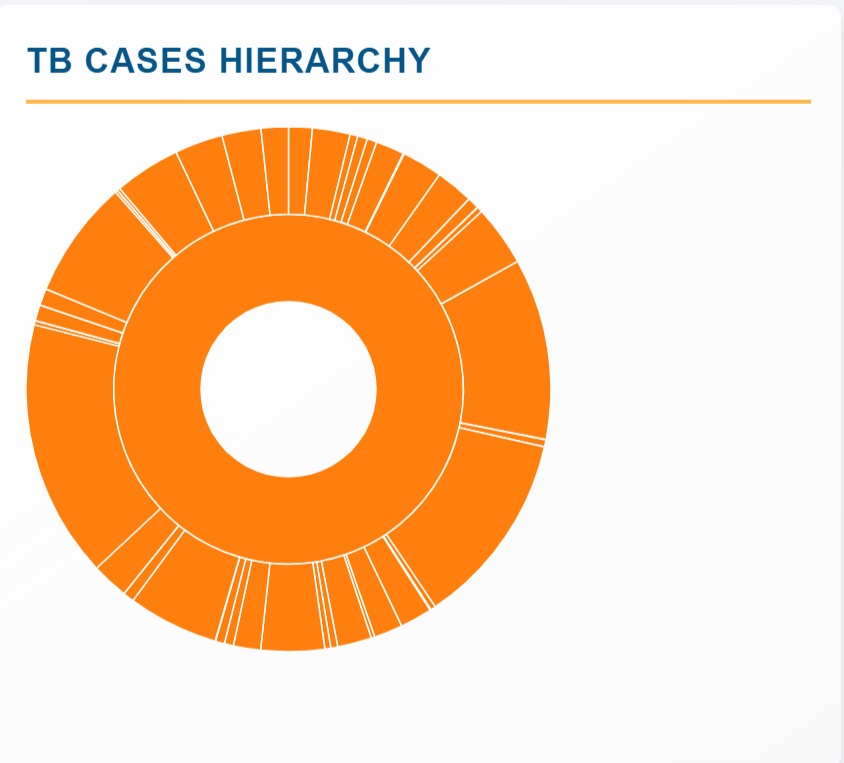
});

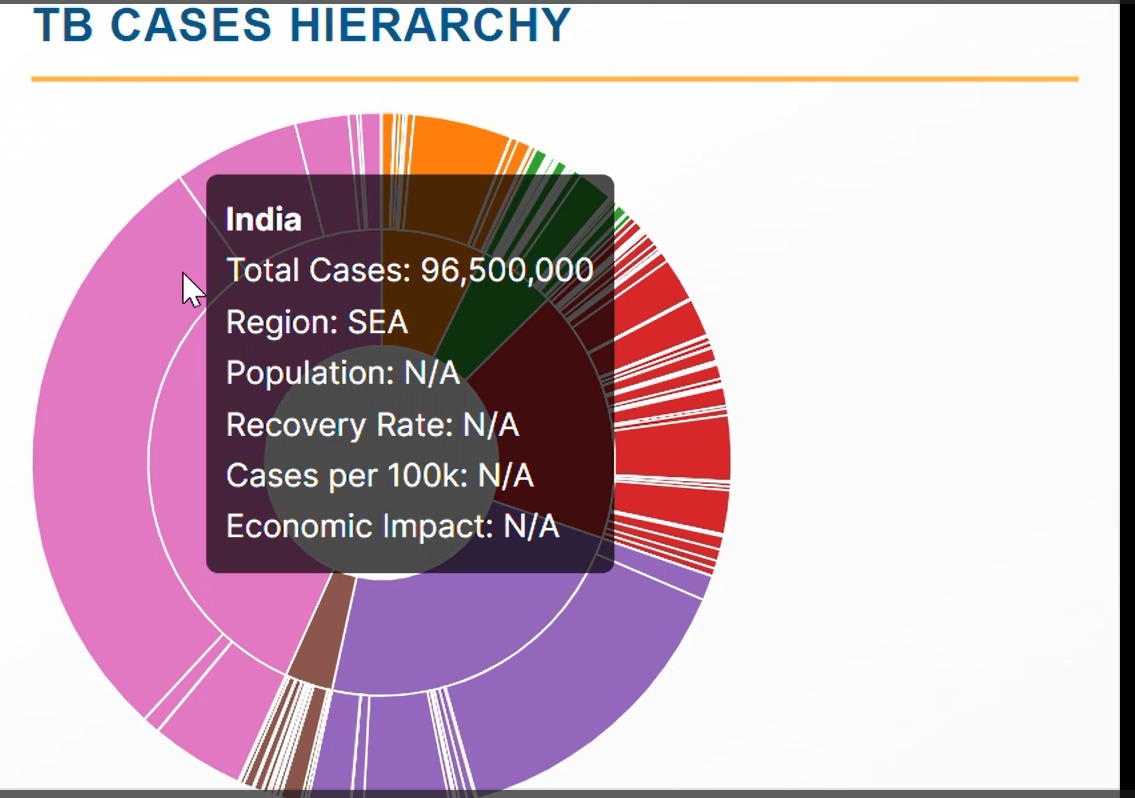
* **Explanation**: The tooltip is hidden initially and made visible when the user hovers over a cell. The tooltip provides detailed information about each country and its TB data.

#### **Step 10: Final Interaction**

On mouseover, the cell is highlighted by adjusting its opacity, and the tooltip shows up with relevant information. When the mouse moves out, the opacity is reset, and the tooltip is hidden.

**5. Sunburst Chart for Hierarchical Data (Bonus Visualization):**

****

****

The goal of this visualization is to create a **Sunburst Chart** that displays the total number of tuberculosis (TB) cases in different countries, grouped by regions. The chart allows users to see the distribution of TB cases across the globe and explore the data interactively, providing insights about the number of cases, population, recovery rate, cases per 100k, and economic impact for each country.

### ****Steps:****

#### **Step 1: Preparing the Data**

We need to transform the input data to create hierarchical structures that will allow the Sunburst chart to work efficiently. The data is grouped by regions and then further subdivided by countries within each region. The total number of TB cases is summed at the country level, and other relevant information (such as population, recovery rate, etc.) is extracted from the dataset.

#### **Code:**

javascript

Copy code

const hierarchyData = d3.rollup(

data,

(v) => d3.sum(v, (d) => +d.total\_cases), // Sum total cases

(d) => d.region, // First level: region

(d) => d.country // Second level: country

);

* **Explanation:**
  + d3.rollup() groups the data by regions and countries, and sums the total TB cases for each country.
  + The v parameter represents the data points for each country.
  + We use the d3.sum() function to calculate the total number of TB cases for each country.

#### **Step 2: Setting up the SVG and Dimensions**

We create an SVG element to hold the Sunburst chart. The SVG is set to a fixed size, and the center of the chart is positioned at the center of the SVG to ensure a symmetrical layout.

#### **Code:**

javascript

Copy code

const svg = d3

.select("#sunburst")

.append("svg")

.attr("width", width)

.attr("height", height)

.append("g")

.attr("transform", `translate(${width / 2},${height / 2})`);

* **Explanation:**
  + An SVG element with a width and height of 350px is appended to the DOM, and a <g> element is added to position the chart's center.

#### **Step 3: Building the Hierarchy**

We construct the hierarchical structure required for the Sunburst chart, which includes regions and their associated countries. The root node will represent the total TB cases, and its children will represent regions and countries.

#### **Code:**

javascript

Copy code

const root = d3

.hierarchy({

name: "TB Cases",

children: Array.from(hierarchyData, ([region, countries]) => ({

name: region,

children: Array.from(countries, ([country, value]) => ({

name: country,

value: value,

})),

})),

})

.sum((d) => d.value || 0);

* **Explanation:**
  + The data is structured in a hierarchical format: the root is "TB Cases", regions are grouped under it, and countries are grouped under each region.
  + The .sum() method is used to assign the total number of TB cases as the value for each node.

#### **Step 4: Defining the Partition Layout**

To create the Sunburst chart, we use a partition layout to convert the hierarchical data into a format that can be drawn as arcs. The partition layout will compute the angles and radii for each node.

#### **Code:**

javascript

Copy code

const partition = d3.partition().size([2 \* Math.PI, radius]);

partition(root);

* **Explanation:**
  + d3.partition() is used to transform the hierarchical data into a format suitable for radial layouts, specifically a Sunburst chart.
  + size([2 \* Math.PI, radius]) specifies that the chart will cover a full circle (360 degrees, or 2π radians), and the radius is defined as the minimum of the width or height divided by 2.

#### **Step 5: Drawing the Arcs**

Each node in the hierarchy is drawn as an arc, with each arc's size representing the total number of TB cases. The d3.arc() function is used to generate the arcs.

#### **Code:**

javascript

Copy code

const arc = d3

.arc()

.startAngle((d) => d.x0)

.endAngle((d) => d.x1)

.innerRadius((d) => d.y0)

.outerRadius((d) => d.y1);

svg

.selectAll("path")

.data(root.descendants())

.enter()

.append("path")

.attr("display", (d) => (d.depth ? null : "none"))

.attr("d", arc)

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

.style("fill", (d) => color((d.children ? d : d.parent).data.name));

* **Explanation:**
  + d3.arc() defines the shape of each arc, with startAngle and endAngle determining the angular span, and innerRadius and outerRadius determining the thickness of the arc.
  + svg.selectAll("path") is used to bind the data and append path elements for each arc.

#### **Step 6: Adding Interactivity**

The Sunburst chart includes interactivity, with tooltips appearing on hover, and the chart zooming into a specific region or country when clicked.

#### **Code:**

javascript

Copy code

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

const countryData = data.find(

(dataPoint) => dataPoint.country === d.data.country

);

d3.select("#tooltip")

.style("display", "block")

.html(`

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

Total Cases: ${d.value.toLocaleString()}<br>

Region: ${d.parent ? d.parent.data.name : "N/A"}<br>

Population: ${countryData ? countryData.total\_population.toLocaleString() : "N/A"}<br>

Recovery Rate: ${countryData ? countryData.recovery\_rate : "N/A"}<br>

Cases per 100k: ${countryData ? countryData.cases\_per\_100k : "N/A"}<br>

Economic Impact: ${countryData ? countryData.economic\_impact : "N/A"}

`)

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

.style("top", event.pageY - 10 + "px");

})

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

d3.select("#tooltip").style("display", "none");

})

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

svg

.transition()

.duration(750)

.tween("scale", () => {

const xd = d3.interpolate(arc.startAngle()(), d.x0);

const yd = d3.interpolate(arc.endAngle()(), d.x1);

const yr = d3.interpolate(arc.innerRadius()(), d.y0);

const yR = d3.interpolate(arc.outerRadius()(), d.y1);

return (t) => {

d3.select(this).attr(

"d",

arc

.startAngle(xd(t))

.endAngle(yd(t))

.innerRadius(yr(t))

.outerRadius(yR(t))

);

};

});

});

* **Explanation:**
  + **Mouseover:** When the user hovers over an arc, a tooltip displays detailed information about the country, including total TB cases, population, recovery rate, cases per 100k, and economic impact.
  + **Mouseout:** Hides the tooltip when the user moves the mouse away.
  + **Click:** Zooms into a specific region or country, focusing on the clicked node by transitioning the angles and radii of the arcs.

#### **Step 7: Tooltip**

The tooltip provides additional information when a user hovers over a segment. It is initially hidden and is only displayed when a user interacts with the Sunburst chart.

#### **Code:**

javascript

Copy code

d3.select("#tooltip").style("display", "none");

* **Explanation:**
  + This ensures that the tooltip is hidden by default and becomes visible when triggered by a hover event on an arc.

**DATA LOADING CODE:**

loadData();

function addExportFunctionality() {

    document.getElementById("export-view").addEventListener("click", () => {

        const svgElements = document.querySelectorAll("svg");

        svgElements.forEach((svg) => {

            const serializer = new XMLSerializer();

            const svgBlob = new Blob([serializer.serializeToString(svg)], {

                type: "image/svg+xml;charset=utf-8",

            });

            const url = URL.createObjectURL(svgBlob);

            const link = document.createElement("a");

            link.href = url;

            link.download = "dashboard-view.svg";

            document.body.appendChild(link);

            link.click();

            document.body.removeChild(link);

        });

    });

**4. Front End:**

#### **1. Overview**

The **Global Tuberculosis (TB) Cases Dashboard** is an interactive visualization tool designed to provide insights into TB cases globally. This dashboard uses **D3.js** and **TopoJSON** to present data in various interactive formats, such as force-directed graphs, geographic distribution maps, sunburst charts, timelines, and tree maps. Users can filter the visualizations by region and year to focus on specific data subsets.

#### **2. Key Features**

* **Region and Year Filters**: Users can filter data based on region and year using dropdown menus.
* **Interactive Visualizations**: Different types of visualizations such as force-directed graphs, maps, sunbursts, and tree maps provide dynamic insights into TB cases globally.
* **Tooltips**: Tooltips are displayed on hover to provide detailed information about specific data points.
* **Responsive Design**: The dashboard adjusts for different screen sizes to ensure usability across devices.

#### **3. Code Snippets & Explanation**

##### **HTML Structure**

The HTML structure defines the layout of the dashboard, including a header, filter section, and multiple visualizations.

html

Copy code

<body>

<div class="dashboard-header">

<div class="dashboard-title">Global Tuberculosis Cases Dashboard</div>

<div class="filter-section">

<div>

<label>Region:</label>

<select id="regionFilter">

<option value="">All Regions</option>

</select>

</div>

<div>

<label>Year:</label>

<select id="yearFilter">

<option value="">All Years</option>

</select>

</div>

</div>

</div>

<!-- Visualization Containers -->

<div class="dashboard-container">

<div class="visualization-large">

<h3>TB Cases Network(Force Directed Graph)</h3>

<div id="force-graph"></div>

</div>

<!-- More visualizations -->

</div>

</body>

* **Dashboard Header**: Includes a title and filter section with dropdowns for selecting a region and year.
* **Visualization Containers**: These are sections where each type of visualization (graph, map, etc.) will be rendered.

##### **CSS Styling**

The CSS styles the dashboard for a professional and responsive look. It defines variables for colors, fonts, and design elements like shadows and borders. Additionally, media queries ensure responsiveness on mobile devices.

css

Copy code

:root {

--primary-color: #065589; /\* Deep Blue \*/

--secondary-color: #5b7064; /\* Muted Olive Green \*/

--background-color: #f4f6f9; /\* Light Gray Background \*/

--card-background: #ffffff; /\* White Background for Cards \*/

--font-family: 'Inter', Arial, sans-serif;

}

body {

font-family: var(--font-family);

background-color: var(--background-color);

padding: 20px;

display: flex;

flex-direction: column;

}

.dashboard-header {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 20px;

background-color: var(--card-background);

padding: 15px;

border-radius: 8px;

box-shadow: rgba(0, 0, 0, 0.1) 0px 6px 12px;

background: linear-gradient(145deg, #ffffff, #f5f5f5); /\* Subtle gradient \*/

}

.visualization-large, .visualization {

background-color: var(--card-background);

border-radius: 8px;

box-shadow: rgba(0, 0, 0, 0.1) 0px 6px 12px;

padding: 20px;

transition: all 0.3s ease;

}

.visualization-large {

grid-column: span 2;

width: 1200px;

height: 800px;

}

* **Variables**: Defined for consistent use of colors and fonts across the design.
* **Flexbox Layout**: Used for the header and visualization container to manage layout effectively.
* **Hover Effects**: Added to visualization cards to provide interactivity, such as hovering to show a shadow effect.

##### **Interactive Visualizations**

The script.js file will contain JavaScript code to render the visualizations using **D3.js**. Below is an example of how to initialize a force-directed graph:

javascript

Copy code

const width = 1200;

const height = 800;

const svg = d3.select("#force-graph").append("svg")

.attr("width", width)

.attr("height", height);

const simulation = d3.forceSimulation()

.force("charge", d3.forceManyBody())

.force("center", d3.forceCenter(width / 2, height / 2));

d3.json("data.json").then(function(data) {

const nodes = data.nodes;

const links = data.links;

simulation.nodes(nodes).on("tick", ticked);

simulation.force("link", d3.forceLink(links).id(d => d.id));

function ticked() {

svg.selectAll("line")

.data(links)

.join("line")

.attr("stroke", "#aaa");

svg.selectAll("circle")

.data(nodes)

.join("circle")

.attr("r", 5)

.attr("fill", "steelblue")

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

.attr("cy", d => d.y);

}

});

* **Force-Directed Graph**: This visualization uses the force layout to display nodes and links representing TB data relationships (e.g., country, region, etc.). The data is loaded from a JSON file and passed to the D3 simulation.

##### **Responsive Design**

The following media query ensures that the dashboard looks good on smaller screens by stacking visualizations vertically:

css

Copy code

@media (max-width: 768px) {

.dashboard-container {

display: flex;

flex-direction: column;

}

.visualization-large {

width: 100%;

}

}

* **Mobile Layout**: The dashboard layout changes to a column-based design when the screen width is 768px or smaller. The large visualization is resized to fit the screen.

**5. CONCLUSIONS:**

In conclusion, this project demonstrates the effective use of D3.js to create an interactive and informative sunburst chart for visualizing tuberculosis (TB) cases across different regions and countries. By leveraging hierarchical data, the chart provides an intuitive, visually appealing representation of the distribution of TB cases, enabling users to explore data by region and country.

The integration of tooltips further enhances the user experience, providing detailed information on key metrics such as total cases, population, recovery rate, cases per 100k, and economic impact. The interactive nature of the chart, including zooming functionality, adds an additional layer of engagement, making it easier for users to navigate complex data. Overall, this project highlights how data visualization tools like D3.js can transform raw data into accessible and meaningful insights, supporting better understanding and decision-making in public health and related fields.