ELECTRIC VEHICLE ADOPTION VS CLEAN ELECTRICITY

BSc – Computer Science
Data Visualization - BST-BCS-13a
SoSe - 2025

Abstract

This project explores the relationship between electric vehicle adoption and grid cleanliness. While global correlations are weak, regional and local analyses reveal adoption hubs and spatial variation driven by policy, market factors, and local conditions. Interactive D3.js and Leaflet visualizations highlight these patterns, demonstrating the value of spatially-informed, interactive approaches for understanding EV adoption.

Jan, Ayesha (3122167)
Kommireddy Umasankari, Anunitha (3122007)
Ibrahim, Maha (3122213)

01 September 2025

Table of Contents

Does Clean Electricity Drive Electric Vehicle Adoption?

Task 1: Topic Selection and Hypothesis

As the global push for decarbonization accelerates, electric vehicles (EVs) are often promoted as a key solution for reducing transport-related emissions. However, the environmental benefits of EVs depend heavily on the cleanliness of the electricity used to charge them. This analysis will explore the relationship between EV adoption (measured by electric cars sold per year) and the carbon intensity of national energy systems (CO_2 emissions per unit of energy consumed).

Hypothesis:

Countries with cleaner electricity, meaning lower CO_2 emissions per unit of energy, will have higher rates of EV adoption. EV use grows faster in countries where electricity comes from clean sources like wind or solar. But if electricity is made from coal or oil, electric cars don't help as much, so fewer people may buy them.

Importance of Investigating this relationship:

- It may show if clean electricity grids are needed before many people start using EVs.
- It can help decide if countries should focus on cleaning their power grids first or promote EVs and clean energy together.
- It adds detail to the conversation about electric transport by looking not just at EV sales, but also at how clean the electricity is.

Task 2: Dataset Research

Search Strategy:

We searched for good and reliable data about energy consumption and electric vehicles (EVs) from trusted websites. We mainly used Kaggle and government open data sites like data.gov. We also checked well-known sources like IEA and Our World in Data and looked for datasets with country-level or regional information on EV use, energy consumption, and CO_2 emissions. We made sure the

datasets cover several years and include the same countries or regions. Finally, we checked that the data can be used for research by looking at their licenses.

Dataset Pairs:

Pair	Dataset 1	Source	Dataset 2	Source
1	Global EV Data Explorer (EV sales & stock)	Our World in Data (IEA-derived)	World energy data 1990 - 2020	Kaggle
2	Global EV Sales: 2010–2024	Kaggle	Primary Energy Consumption per Capita	Our World in Data
3	Number of New Electric Cars Sold	Our World in Data (IEA-derived)	Energy Consumption by Source & Country	Our World in Data
4	Global EV sales 2010-2024	Kaggle	Global Fossil Fuel Consumption	Our World in Data
5	Electric Vehicle Population Dataset	Kaggle	Global Energy Consumption	Kaggle

Task 3: Data Quality Assessment

Pair	Dataset Pair	Completeness	Temporal Alignment	Geographic Alignment
1	Global EV Data Explorer IEA / World energy data 1990 - 2020	High – Minimal missing values	High – Multi-year, annual	High – Global coverage
2	Global EV Sales 2010–2024 Kaggle / Primary Energy Consumption per Capita Our World in Data	High – Mostly complete	Medium–High – Overlap exists	High – Global coverage
3	Number of New Electric Cars Sold — OWID / Energy Consumption by Source & Country OWID	High – Both clean datasets	High – Perfect overlap	High – Same country codes
4	Electric Vehicle Population Dataset Kaggle / Global Energy Consumption Kaggle	Medium – Missing entries	Medium – Limited overlap	Medium – EV dataset less global
5	Global EV Sales 2010–2024 Kaggle / Global Fossil Fuel Consumption — OWID	High – Minimal missing values	High – Overlapping years possible	High – Broad country coverage

Pair	Data Granularity	Source Reliability	Overall Suitability	
1	Annual, aggregated by country	Very High (IEA gold standard, Kaggle credible)	High	Perfect for EV adoption vs energy/emissions; clean and well-aligned
2	Annual per country	High (Kaggle + OWID credible)	High	Great for long-term trend comparisons; minor range differences
3	Annual per country	Very High (OWID)	High	Easy merge, lines up well, but sources are same
4	Annual but inconsistent for EV dataset	Medium (user-sourced Kaggle datasets)	Medium	Requires cleaning; limited alignment and completeness
5	Annual per country	High	High	Strong for EV adoption vs fossil fuel decline comparisons

Chosen Datasets:

1. International Energy Agency. Global EV Outlook 2025. – processed by Our World in Data
2. World energy data 1990 - 2020 – Kaggle

Task 4: Identify Common Dimensions

Common Dimensions:

- Temporal dimension: Both datasets include year (first dataset: Year, second dataset: Year), enabling alignment across time periods.

- Geographic dimension: Both datasets have country identifiers (first dataset: country, second dataset: Entity/Code) for matching countries.

- Numerical dimension:
  - Dataset 1 contains energy and emissions metrics (CO2 emissions from fuel combustion, energy production/consumption, renewables/electricity shares, oil/gas/coal production and consumption, energy intensity, etc.).
  - Dataset 2 contains EV adoption metrics (Electric cars sold).

Possible Connections:

- Temporal connection: Match each country's energy and emissions indicators with its EV sales in the same year.

- Geographic connection: Link energy/emissions data for each country with the corresponding EV sales data.

- Numerical connection: Compare EV adoption trends with factors such as fossil fuel consumption, CO_2 emissions, renewable energy shares, and electricity production/consumption.

Documented Join Keys:

- Country ↔ Entity
- Year ↔ Year

Data Linkage Strategy:

- Make sure country names/codes match in both datasets (for example, "United States" vs "USA").
- Check that the Year format is the same in both datasets and only use years that exist in both.
- Create a carbon intensity value for each country and year.
- Join the datasets using (Country/Entity, Year) so that EV sales line up with the energy and emissions data.
- Include energy breakdowns like coal, oil, gas, renewables, and wind/solar shares to compare clean vs dirty energy.
- After merging, look for missing values or mismatches and fix them (by filling in or removing).
- In the final table, each row will represent one country-year and will have:
  - Electric cars sold
  - Carbon intensity
  - Clean Electricity
  - Energy source fractions (coal, oil, gas, renewables, wind/solar)
  - Other energy/emission info (like electricity use, energy intensity)

Task 5: Exploratory Data Analysis

The goal of this task was to perform an initial exploration of the merged EV and CO_2 emissions datasets to uncover patterns, outliers, and potential relationships. This step helps guide the final visualization and confirms whether our hypothesis about cleaner electricity and EV adoption holds across countries and years.

Results:

- Line Chart: Shows the trend of electric cars sold from 2015 onwards for the top 10 countries. This allows us to see which countries are increasing EV adoption fastest. Hover interactions highlight individual country trends.

- Scatter Plot: Plotted EV sales against CO_2 emissions from fuel combustion for the same countries. Outliers (countries with unusually high EV sales or emissions) are highlighted in red. Brushing allows selection of a subset of points to focus on specific ranges or countries.

- Map Visualization: Used Leaflet to show geographic distribution of total EVs for the top 10 countries. Marker size reflects total EV sales, and popups display country-specific totals. Color coding matches the line/scatter charts for consistency.

Outlier Analysis:

- Identified the top 5% of EV sales and CO_2 emission values as potential outliers.
- These points were emphasized in the scatter plot to check for unusual patterns or exceptions in the relationship between EV adoption and national emissions.

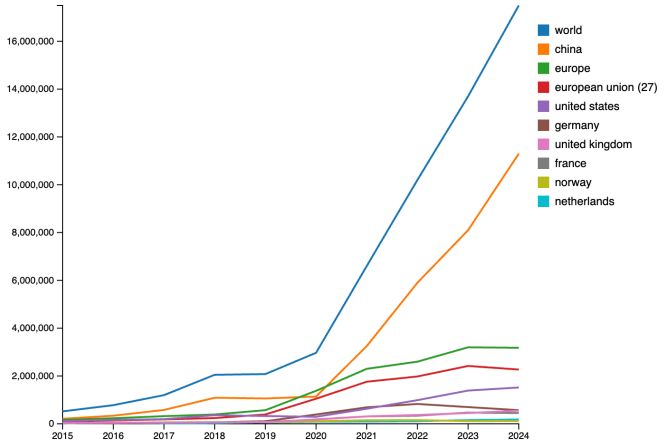
Initial Insights:

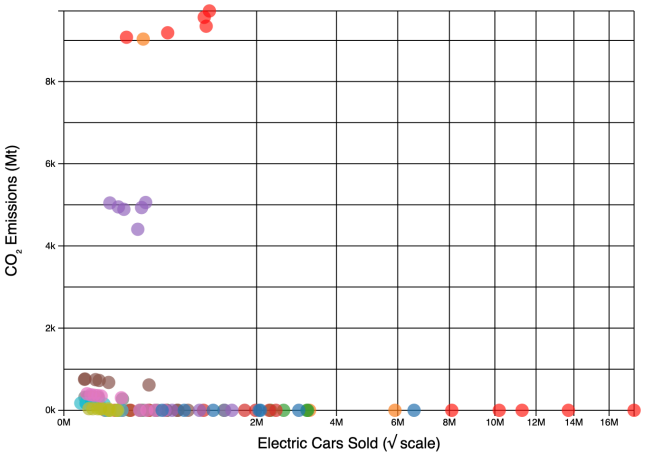
- Countries with cleaner electricity grids do not always have the highest EV sales, suggesting additional factors influence adoption.
- Some high-EV countries also have high CO_2 emissions, indicating that EV adoption may be driven by policy or incentives rather than electricity cleanliness alone.
- Line charts confirm that growth trends differ: some countries show steady adoption, while others spike in recent years.

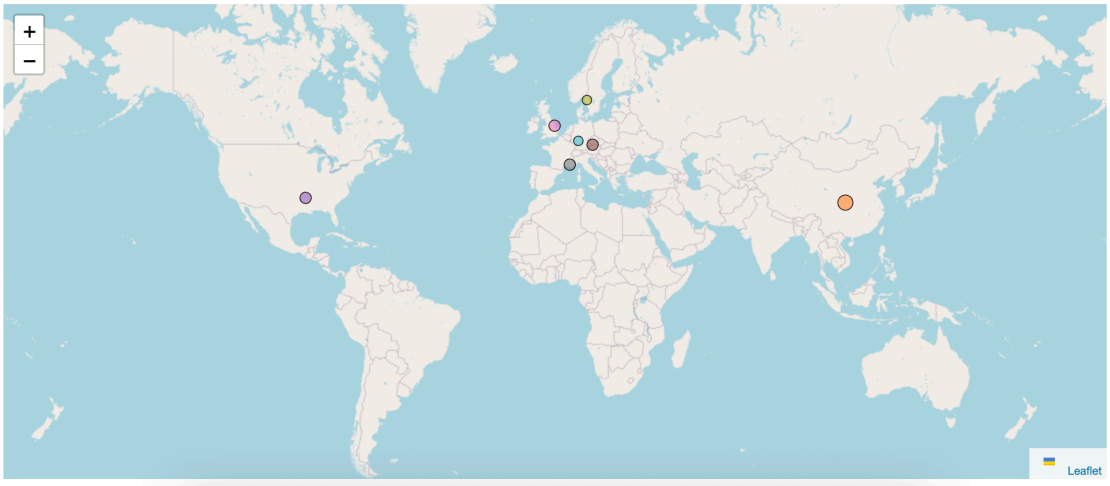
Potential Correlations:

- A preliminary Pearson correlation coefficient was calculated between EV sales and CO_2 emissions, giving an initial quantitative measure of the relationship.
- While the correlation is weakly negative in some years (supporting the hypothesis), it is not uniform across all countries, highlighting the complexity of adoption patterns.

EV Exploratory Analysis (Top 10 Countries)







Task 6: Choose Visualization Type

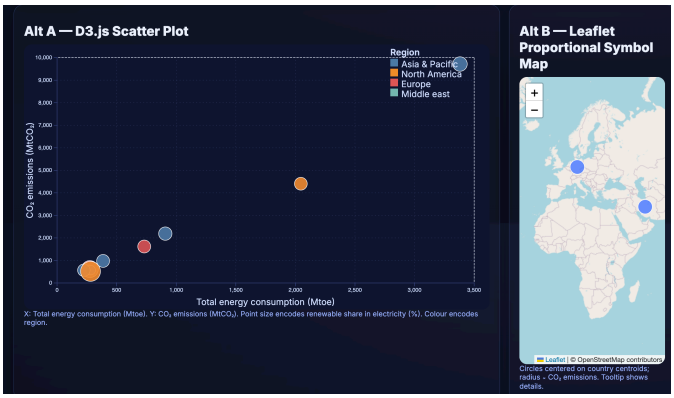
Justification and Design Choices:

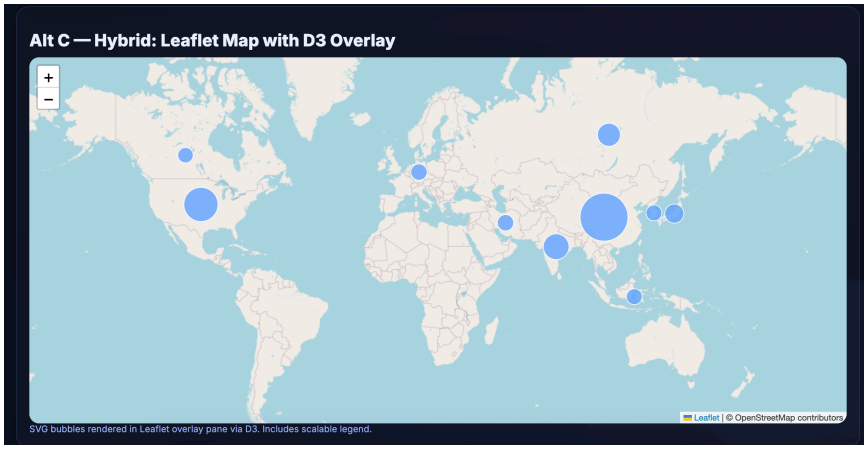
- Spatial vs. non-spatial: The scatter plot is best for revealing non-spatial relationships (EV adoption vs. electricity cleanliness), which directly addresses our hypothesis. The map-based views provide important spatial context, showing where adoption is happening geographically.

- Data volume & complexity: With only 10 countries, all three visualizations remain clear and readable. The D3 scatter plot scales better for more variables (e.g., adding color = region, size = renewables share). Maps remain effective with low density, without requiring clustering.

- Audience needs:
  - Executives benefit from a map overview for quick geographic patterns.
  - Analysts need a scatter plot for precise quantitative comparisons, tooltips, and regression checks.
  - Educators and general audiences may prefer the hybrid view, which demonstrates how geographic and multivariate data can be layered together.

Final Choice: For the visualization, we chose both the D3 scatter plot and the Leaflet map. The D3 scatter plot because it most directly addresses our research question: Does clean electricity drive EV adoption? It allows us to show the statistical relationship between EV sales and carbon intensity, while still supporting interactivity (tooltips, brushing, highlighting). The map visualizations will be used as supporting context in the storytelling phase to provide a geographical perspective.





Task 7: Data Preprocessing

Cleaning and Standardizing:

- Convert 'Year' to integer: ensures numeric comparison and filtering.
- Convert numeric columns to floats: guarantees all computations work correctly.
- Handle missing values:
  - Any missing or invalid numeric values are replaced with 0.
  - Ensures no calculation errors when computing derived metrics.
- Align temporal coverage:
  - Only keep years that exist in both datasets.
  - Ensures valid year-to-year comparisons.
- Normalize numeric variables for visualization:
  - EV_norm = Electric cars sold ÷ max(Electric cars sold)
  - CleanElec_norm = Clean electricity share ÷ max(clean electricity share)
  - Useful for D3.js visualizations.

```
function cleanData(evData, energyData) {
      //  Convert Years to numbers
      evData.forEach(d => d.Year = +d.Year || 0);
      energyData.forEach(d => d.Year = +d.Year || 0);

      //  Standardize numeric columns
      const numericCols = energyData.columns.filter(c => !["country","Region"].includes(c));
      energyData.forEach(d => {
        numericCols.forEach(col => {
          d[col] = parseFloat(d[col]) || 0;
        });
      });
```

The cleanData function ensures numeric consistency in two datasets. It converts the Year field to numbers, defaulting to 0 if invalid. It also converts all non-text columns in energyData to floats, setting invalid values to 0.

Derived Calculations:

1. Carbon intensity (CO_2 / energy): Carbon Intensity (MtCO2 per EJ) = CO_2 emissions from fuel combustion ($MtCO_2$) ÷ Total energy consumption (EJ)
2. Clean electricity share (%):
   - Sum of renewable electricity and wind/solar share.
   - Captures the "cleanliness" of the electricity grid.

Energy source fractions:

3. Coal fraction (%) = Coal consumption ÷ Total energy consumption
4. Oil fraction (%) = Oil consumption ÷ Total energy consumption
5. Gas fraction (%) = Gas consumption ÷ Total energy consumption
6. Renewables fraction (%) = Share of renewables
7. Wind/Solar fraction (%) = Share of wind and solar

Renaming Columns:

- EV dataset column Entity → country
- Ensures consistent join key between datasets.

Task 8: Join Datasets

Strategy:

- Join datasets on (country, year).
- Temporal + key-based join
- Use d3.group() to create a map of energy data keyed by country + "_" + year.
- Merge corresponding EV records with energy metrics.
- Handle edge cases: only include rows where both datasets have matching year and country.
- Validation:
  - Ensure no missing values in merged dataset.
  - Verify numeric ranges (e.g., EV sales > 0, electricity share 0–100%).

```
function mergeDatasets(evData, energyData) {
      const energyMap = d3.group(energyData, d => d.country + "_" + d.Year);
      const merged = [];

      evData.forEach(ev => {
        const key = ev.country + "_" + ev.Year;
        const energy = energyMap.get(key);
        if (energy) {
          merged.push({ ...ev, ...energy[0] });
        }
      });

      return merged;
    }
```

This function combines the EV dataset with the energy dataset based on matching country and Year values.

- Index energy data: Creates a lookup table (Map) where the key is "country_Year" and the value is the corresponding energy records.
- Iterate over EV data: For each EV record, it builds the same "country_Year" key and checks if a matching energy record exists.
- Merge records: If a match is found, it merges the EV data with the first energy record for that key and adds it to the merged array.
- Return merged dataset: Produces a unified dataset where each entry contains both EV and energy attributes for the same country and year.

Outcome:

masterData: each row represents one country-year with:

- Electric cars sold
- Carbon intensity
- Energy source fractions (coal, oil, gas, renewables, wind/solar)Energy/emission info (electricity consumption, energy intensity, etc.)

Preparing GeoJSON for Leaflet:

Purpose:

- Leaflet requires a geographic format to plot country points.
- Each feature includes both geometry and properties.

Implementation:

- Geometry:
  - type: "Point"
  - Coordinates [longitude, latitude]
- Properties:
  - country, year, EV, EV_norm, CleanElec, CleanElec_norm, CarbonIntensity, Energy source fractions: coal, oil, gas, renewables, wind/solar

Result:

master_data_geoJSON: can be directly used to plot interactive maps, showing EV adoption and clean electricity at the country level.

Task 9: Statistical Analysis

Statistics Measured:

- Global correlations
  - EV vs Carbon Intensity: +0.170 (weak positive)
  - EV vs Clean Electricity: –0.069 (near zero / very weak negative)
- Regional correlations
  - Asia & Pacific: +0.257 (weak positive)
  - Europe: –0.175 (weak negative)
  - South/Latin America: +0.115 (very weak positive)
  - North America: +0.427 (moderate positive)
- Regression (non-spatial)
  - The scatter plot of EV sales (thousands) vs Carbon Intensity ($MtCO_2/EJ$) shows a slightly positive slope, consistent with the weak global positive correlation.
- Spatial statistics (Leaflet clustering)
  - Marker clustering highlights dense pockets of EV activity in Europe, East Asia, and parts of North America.
- Local/Geographically Weighted Regression (GWR demo)
  - Clicking different areas computes a local slope (weighted by distance). Local slopes change sign and magnitude, confirming spatial heterogeneity in the EV–cleanliness relationship.

```
// ============================
 // Correlation Coeffiecient
 // ============================

 let ev = masterData.map(d => d.EV);
 let ci = masterData.map(d => d.CarbonIntensity);
 let clean = masterData.map(d => d.CleanElec);

 let corrEV_CI = ss.sampleCorrelation(ev, ci);
 let corrEV_Clean = ss.sampleCorrelation(ev, clean);
```

```
  d3.select("#stats").append("p").text(`Correlation EV vs Carbon Intensity:
${corrEV_CI.toFixed(3)}`);
  d3.select("#stats").append("p").text(`Correlation EV vs Clean Electricity:
${corrEV_Clean.toFixed(3)}`);
```

This code calculates and displays correlation values between EV adoption and energy-related metrics:

- Extract data: It pulls out arrays of EV, CarbonIntensity, and CleanElec values from masterData.
- Compute correlations: Using the simple-statistics library (ss.sampleCorrelation), it calculates the correlation coefficient between:
    - EVs and Carbon Intensity
    - EVs and Clean Electricity
- Display results: It appends two <p> elements inside the #stats container with the correlation values, formatted to 3 decimal places.

This measures how strongly EV growth relates to carbon intensity and clean electricity use, then shows the results on the webpage.

```
// Regression line
                let points = masterData.map(d => [d.CarbonIntensity, d.EV_perThousand]);
                let lr = ss.linearRegression(points);
                let lrLine = ss.linearRegressionLine(lr);

                let xRange = d3.extent(masterData, d => d.CarbonIntensity);
                let yRange = xRange.map(xv => lrLine(xv));

                g.append("line")
                .attr("x1", x(xRange[0])).attr("y1", y(yRange[0]))
                .attr("x2", x(xRange[1])).attr("y2", y(yRange[1]))
                .attr("stroke", "red").attr("stroke-width", 2);
```

This code computes a best-fit regression line for the relationship between Carbon Intensity and EV adoption per thousand, then draws it on the chart.

- Prepare data for regression: Creates coordinate pairs [x, y] where x = CarbonIntensity and y = EVs per 1000 people.
- Fit a regression model: Uses simple-statistics to calculate the best-fit regression line.
    - lr stores the slope & intercept.
    - lrLine(x) gives the predicted y value for any x.
- Find the line's endpoints: Gets the min/max carbon intensity (xRange) and computes the corresponding predicted EV values (yRange).
- Draw the regression line: Plots a red line from the predicted y at the lowest CarbonIntensity to the predicted y at the highest.

Results:

- No strong global link
    - The hypothesis "cleaner electricity → more EV adoption" is not supported globally by these simple bivariate measures.

    - The near-zero global correlation with Clean Electricity (–0.069) suggests that grid cleanliness alone does not explain EV sales.
    - The weak positive global correlation with Carbon Intensity (+0.170) runs counter to the hypothesis, and likely reflects confounding factors (market size, income, industrial policy, incentives, domestic manufacturing, charging roll-out).

- Regional differences matter
    - Europe (–0.175) trends negative (cleaner → more EVs), which aligns with the hypothesis, but the effect is small.
    - North America (+0.427) shows a moderate positive link (higher carbon intensity with higher EV sales), implying policy/market factors dominate cleanliness (e.g., generous incentives, model availability, consumer preferences).
    - Asia & Pacific (+0.257) is mixed: large markets with significant EV sales may still rely partly on fossil-heavy grids.

- Spatial clustering shows real hubs
    - The map's clusters reveal concentrated EV adoption in Europe, China/East Asia, and North America. These hubs correspond to policy leadership, supply chains, and consumer incentives, not just grid cleanliness.

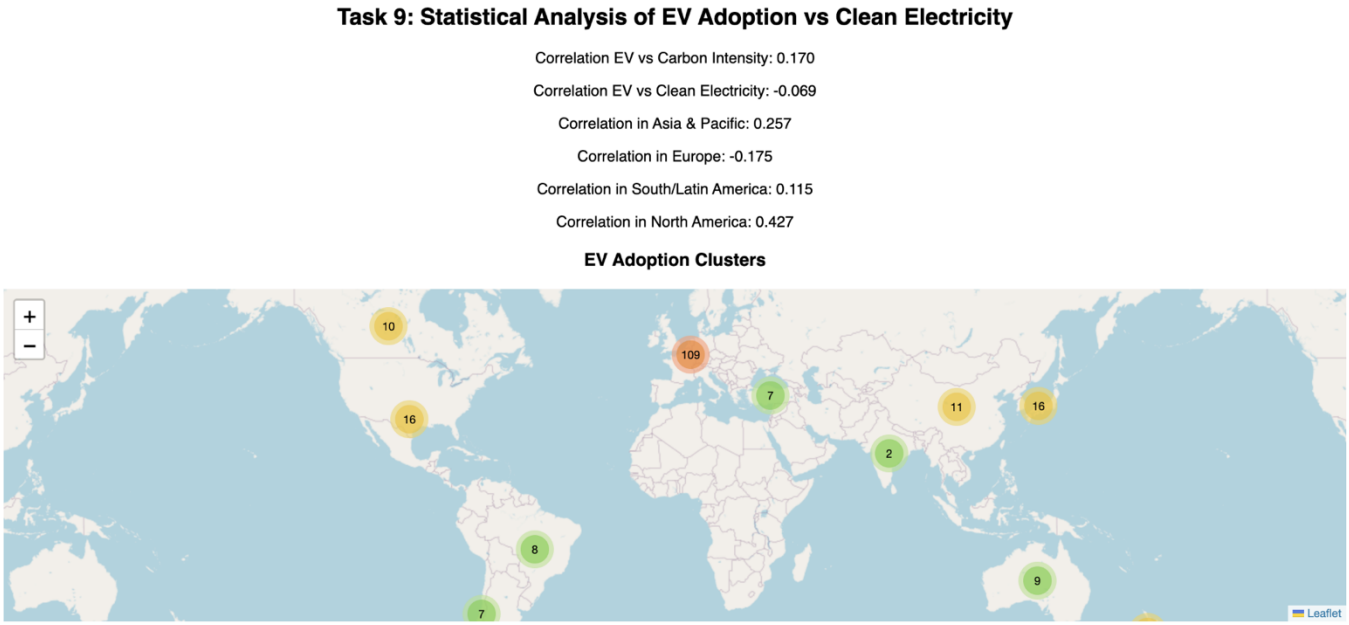
- Local relationships vary (GWR)
    - The local (weighted) regression reacts to the clicked location:
    - In parts of Europe/Nordics, local slopes often tilt negative (cleaner grids associate with higher EV sales).
    - Around North America/East Asia, slopes can turn positive or weak, reflecting economic scale and policy overshadowing grid composition.
    - This non-stationarity explains why global averages blur the story.

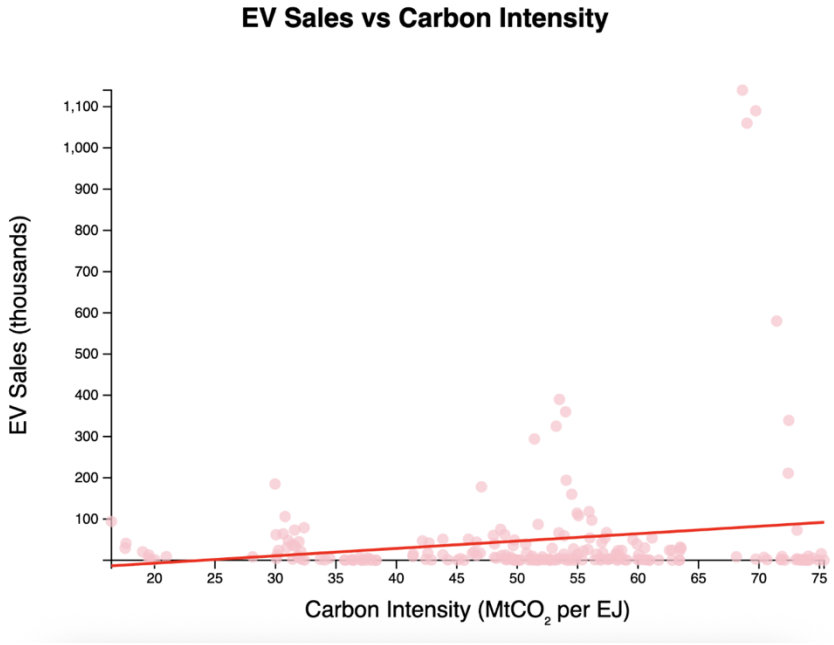
Patterns and Anomalies:

- Probable outliers:
    - Norway/others in Nordics: very high EV sales with low carbon intensity (supports hypothesis and stands out in the plot).
    - China/US: high EV sales despite moderate–higher carbon intensity, illustrating policy, market size, and supply chains can dominate.
- Clusters in Western Europe and coastal China reflect dense EV ecosystems (charging, incentives, model variety).

Spurious Correlations:

- Scale effects: Using absolute EV sales favors large populations/economies. Per-capita or per-driver measures would reduce this bias.
- Timing: EV uptake today may respond to policies and grid changes from prior years (lag effects). Same-year correlation can understate true relationships.
- Omitted variables: GDP per capita, subsidies, fuel prices, urbanization, charging density, domestic production are not included and can flip signs.
- Data mix: If multiple years are pooled, within-country trends and between-country differences get conflated.





Task 10: Basic Visualization Structure

Overview:

This template provides a basic visualization structure for EV data using D3.js and Leaflet, combining non-spatial (scatter plot) and spatial (map) visualizations. It implements a responsive SVG container, scales and axes setup, margin conventions, a loading indicator, and proper Leaflet map initialization.

Responsive SVG Container:

```
const svg = d3.select("#chart")
                .append("svg")
                .attr("viewBox", `0 0 ${chartWidth} ${chartHeight}`)
                .attr("preserveAspectRatio", "xMidYMid meet")
```

```
            .append("g")
            .attr("transform", `translate(${margin.left},${margin.top})`);
```

- viewBox ensures the SVG scales responsively with the container width.
- preserveAspectRatio maintains proportions when resizing.
- The inner <g> element is translated according to margins to avoid overlapping axes.

Margin Conventions:

```
const margin = {top: 30, right: 100, bottom: 50, left: 80};
const innerW = chartWidth - margin.left - margin.right;
const innerH = chartHeight - margin.top - margin.bottom;
```

- Standard margin conventions are applied to provide space for axes, labels, and titles.
- Prevents chart content from overlapping axes or edges of the container.

Scales Setup:

```
const xScale = d3.scaleLinear().range([0, innerW]);
const yScale = d3.scaleLinear().range([innerH, 0]);
cont sizeScale = d3.scaleSqrt().range([3, 20]);
const colorScale = d3.scaleSequential(d3.interpolateRgb("pink", "steelblue"));
```

- Linear scales map raw data to pixel positions.
- xScale: maps Clean Electricity Share (%) to horizontal pixel positions.
- yScale: maps Electric Cars Sold to vertical positions.
- sizeScale: circle radius grows with data values (sqrt ensures area scaling).
- colorScale: encodes values using a gradient from pink → steelblue.
- Scales are updated dynamically once data is loaded to accommodate the full data range.

Axes Groups:

```
svg.append("g").attr("class","x-axis").attr("transform", `translate(0, ${innerH})`);
svg.append("g").attr("class","y-axis");
```

Axes are separate <g> elements to allow independent updates when data changes.
X-axis positioned at the bottom; Y-axis at the left.

Axis Labels:

```
        svg.append("text")
            .attr("class","x-label")
            .attr("x", innerW/2)
            .attr("y", innerH + 48)
            .attr("text-anchor","middle").
            text("Clean Electricity Share (%)");

        svg.append("text")
            .attr("class","y-label")
            .attr("x", -innerH / 2)
            .attr("y", -60)
            .attr("transform", "rotate(-90)")
```

```
                .attr("text-anchor", "middle")
                .text("Electric Cars Sold");
```

- Clear labeling enhances interpretability, so users know what each axis represents.
- X-axis label is centered below the chart.
- Y-axis label is rotated, centered vertically, and aligned left.

Loading Indicator:

```
<div class="loading" id="loading">Loading data...</div>
```

- Provides visual feedback to the user while data is being loaded.
- Improves user experience for datasets that take time to fetch.

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

- Hides the indicator once the data is successfully loaded.

D3 Scatter Plot Basic Selection:

```
const circles = svg.selectAll("circle").data(data, d => d ? (d.Country + d.Year) : this);
.merge(circles)
.transition().duration(500)
.attr("cx", d => xScale(d.CleanElec))
.attr("cy", d => yScale(d.EV))
.attr("r", d => sizeScale(d.EV))
.attr("fill", d => colorScale(d.CleanElec));
```

- Demonstrates basic D3.js selection and data binding.
- Each data point is represented as a circle, showing the relationship between Clean Electricity (%) and EV Sales.
- The radius and color are chosen for visibility and simplicity.

Leaflet Map Initialization:

```
const map = L.map('map', { center: [20, 0], zoom: window.innerWidth < 768 ? 1 : 2, zoomControl:
true, dragging: true, tap: true, scrollWheelZoom: false });
            L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
                maxZoom: 18,
                attribution: '&copy; OpenStreetMap contributors'
            }).addTo(map);
```

- Initializes a global view of EV data for spatial visualization. The map is centered at latitude 20, longitude 0, giving a world overview.

- Zoom:
  - If the screen is small (<768px width): zoom level 1 (fits more of the world).
  - Otherwise: zoom level 2 (gives a closer world overview).

- Controls & Interactions:
  - zoomControl: true → shows zoom buttons.
  - dragging: true → allows panning by dragging.
  - tap: true → optimizes touch interactions for mobile.

- scrollWheelZoom: false → disables zooming by scroll to prevent accidental zooming when scrolling the page.

- OpenStreetMap tiles provide a lightweight base map without requiring proprietary APIs.

Component Structure Planning:

- : Leaflet map for spatial insights.
- : D3 scatter plot for non-spatial statistical analysis.
- Loading indicator placed above visualizations ensures user feedback.
- CSS ensures visual alignment between map and scatter plot, keeping a clean, centered layout.

Task 10: Basic Visualization Structure

Task 11: Implement Main Visualization

D3 Scatter Plot:

Enter/Update/Exit Pattern:

```
const circles = svg.selectAll("circle").data(data, d => d.Country + d.Year);

circles.enter().append("circle")
    .attr("opacity", 0.9)
    .merge(circles)
    .transition().duration(500)
    .attr("cx", d => xScale(d.CleanElec))
    .attr("cy", d => yScale(d.EV))
    .attr("r", d => sizeScale(d.EV))
    .attr("fill", d => colorScale(d.CleanElec));

circles.exit().remove();
}
```

Implements the D3 data join pattern to dynamically create, update, and remove circles based on the selected year.

Scales/Axes Shapes:
Explained above in task 10.

Leaflet Map:

```
data.forEach(d => {
     const marker = L.circleMarker([d.lat, d.lon], {
     radius: 4 + sizeScale(d.EV)/2,
     fillColor: colorScale(d.CleanElec),
     color: "#333",
     weight: 1,
     fillOpacity: 0.9
 }).bindPopup(`
 <b>${d.Country} (${d.Year})</b><br>
 EV Sales: ${d.EV.toLocaleString()}<br>
 Clean Electricity: ${(d.CleanElec).toFixed(2)}%
 `);
```

- Loops through each entry in the filtered dataset.
- Each d object represents a country's EV and clean electricity data for a specific year.
- This ensures every country gets a marker on the map.

Custom Markers:
- L.circleMarker creates a circular marker at the coordinates [latitude, longitude].
- Different from a default Leaflet marker (which uses an icon), a circle marker:
    - Can scale in size dynamically.
    - Can change color based on data values.
    - Integrates smoothly with clustering and thematic styling.

Colour Scheme:
- Uses a D3 color scale to map clean electricity share to a color gradient:
    - Lower clean electricity: pink.
    - Higher clean electricity: steelblue.
- Gives immediate visual sense of electricity cleanliness per country.

Popups:
- Displays country name, year, EV sales, and clean electricity share.
- Formats numbers with commas (e.g., 1,234).
- Ensures percentages show two decimal places.
- Users see detailed info on click, enhancing interactivity.

Marker Clustering:
- Combines nearby markers into a single cluster icon.
- Shows the number of markers in that area.
- Reduces visual clutter on a global map.
- Improves performance when displaying many countries.

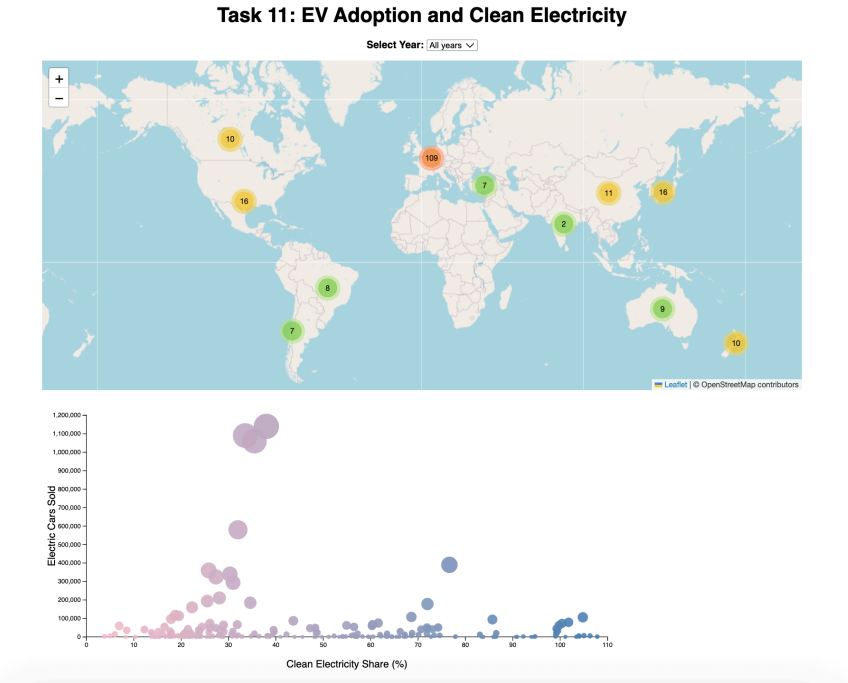
Modularity and Extensibility:

- updateChart(data) and updateMap(data) are separate, reusable functions.

- Easy to add new data dimensions, filters, or map features.
- Data-driven design allows scaling to larger datasets.

Performance Considerations:

- Marker clustering improves map performance for many markers.
- Smooth D3 transitions improve visual experience without blocking rendering.
- Dropdown filtering avoids reloading all data; it filters in memory.



Task 12: Add Interactivity

D3.js Interactivity:

Zoom (d3-zoom):

```
const zoom = d3.zoom()
                .scaleExtent([0.5, 10])
                .translateExtent([[0, 0], [chartWidth, chartHeight]])
                .on("zoom", throttle(zoomed, 16));

d3.select("svg").call(zoom);
```

- Allows users to pan and zoom the scatter plot.
- scaleExtent limits how far the user can zoom in or out.
- translateExtent restricts the panning area to the SVG canvas.

Tooltips:

```
circles.enter().append("circle")
                        .attr("opacity", 0.9)
                        .on("mouseover", (event, d) => {
                            tooltip.style("display","block")
                            .html(`<b>${d.Country}</b><br>EV: ${d.EV.toLocaleString()}<br>Clean
Elec: ${(d.CleanElec).toFixed(2)}%`);
                            // Highlight map marker
```

```
                            const key = d.Country + d.Year;
                            if (markerMap[key]) markerMap[key].setStyle({color:"yellow",
weight:3});
                        })
                        .on("mousemove", event => {
                            tooltip.style("left", (event.pageX+10)+"px")
                                .style("top", (event.pageY-20)+"px");
                        })
                        .on("mouseout", (event, d) => {
                            tooltip.style("display","none");
                            const key = d.Country + d.Year;
                            if (markerMap[key]) markerMap[key].setStyle({color:"#333",
weight:1});
                        })
```

- Hovering over a circle shows a tooltip with detailed data.
- Highlights the corresponding Leaflet marker to provide linked visual feedback.
- mouseover: shows tooltip with country, EV sales, and clean electricity.
- mousemove: positions tooltip near cursor.
- mouseout: hides tooltip.

Brush:

```
const brush = d3.brush()
                    .extent([[0,0], [innerW, innerH]])
                    .on("end", throttle(brushed, 16));

svg.append("g").attr("class", "brush").call(brush);
```

d3.brush() enables click-and-drag selection on the scatter plot.

```
function brushed({selection}) {
                    if (!selection) {
                        updateMap(currentData);
                        return;
                    }
                    const [[x0,y0],[x1,y1]] = selection;
                    const selected = currentData.filter(d => {
                        const cx = xScale(d.CleanElec);
                        const cy = yScale(d.EV);
                        return cx >= x0 && cx <= x1 && cy >= y0 && cy <= y1;
                    });
updateMap(selected);
```

- This function is called whenever the user draws or adjusts a brush (selection rectangle) on the scatter plot.
- {selection} is an object provided by D3 that contains the pixel coordinates of the brush rectangle.
- Filters scatter plot points within the brush area.
- Updates Leaflet map to show only the selected markers (cross-filtering).
- Calls a function (updateMap) to update another visualization (likely a Leaflet map) based on the points selected in the scatter plot.
- This implements linked brushing, connecting the scatter plot with a spatial view.

Click Interaction (Linked Views):

```
.on("click", (event, d) => {
                            const latlng = L.latLng(d.lat, d.lon);
                            map.flyTo(latlng, 4, { animate: true, duration: 1 });
                            const marker = markerMap[d.Country + d.Year];
                            if (marker) marker.openPopup();
                        })
```

- Clicking a circle:
  - Animates map to country location (flyTo).
  - Opens the corresponding Leaflet marker popup.
- Implements linked views between chart and map.

Leaflet Interactivity:

Popups and marker clustering already documented above in Task 11.

Cross-Filtering:

- Implemented via dropdowns and brush on the scatter plot.
- Year/Region dropdowns update both:
  - Scatter plot (D3)
  - Map markers (Leaflet)
- Brush selection on the scatter plot filters the map markers to only show the selected countries.
- Ensures both visualizations always reflect the same subset of data.
- Users can explore correlations interactively (e.g., selecting a region or a set of countries) and immediately see results on the map.

```
function filterData() {
                    const year = yearSelect.property("value");
                    const region = regionSelect.property("value");

                    let filtered = masterData;

                    if (year !== "All years") filtered = filtered.filter(d => d.Year === +year);
                    if (region !== "All regions") filtered = filtered.filter(d => d.Region ===
region);

                    currentData = filtered;
                    // Reset zoom on filtering
                    d3.select("svg").transition().duration(500).call(zoom.transform,
d3.zoomIdentity);
                }
```

- Retrieves the currently selected year and region from UI elements (<select> dropdowns).
- Starts with the full dataset.
- Applies filters conditionally:
  - Year: Only include records matching the selected year (converts to number with +year).
  - Region: Only include records matching the selected region.

- Produces a filtered array of data points and stores the filtered dataset in currentData for use by chart and map updates.
- Calls update functions to redraw the scatter plot and Leaflet map with the filtered data.
- Resets the scatter plot zoom/pan to default view using a smooth 500ms transition.
- Ensures the filtered data is fully visible regardless of previous zoom state.

Smooth transitions and animations:

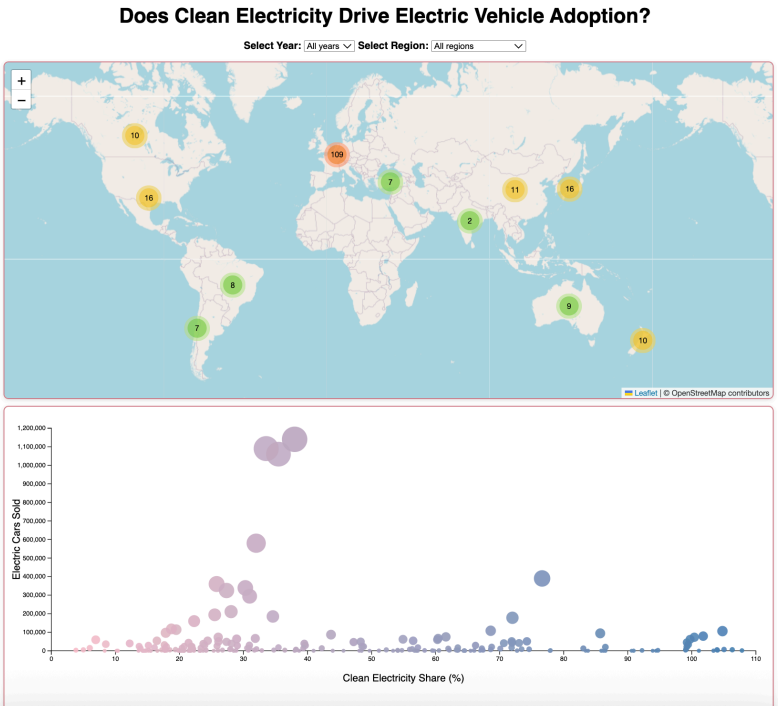
```
svg.select(".x-axis").transition().duration(500).call(d3.axisBottom(xScale));
```

```
map.flyTo(latlng, 4, { animate: true, duration: 1 });
```

- Transitions:
  - Smoothly move circles to new positions when filtering or zooming.
  - Update axes with animation for continuous visual flow.
- Leaflet map uses map.flyTo() to animate movement to a selected country when a circle is clicked.
- Enhances user experience by avoiding sudden jumps and making interactions feel fluid and responsive.

Visual feedback for user actions:

- Provides immediate cues that interactions are happening.
- Hover Feedback:
  - Highlights scatter plot circles and corresponding map markers when hovered.
  - Shows tooltip with detailed info.
- Click Feedback:
  - Animates the map to the country's location.
  - Opens marker popup to provide immediate confirmation of the selection.
- Brush Feedback
  - Selecting points on the scatter plot filters map markers, giving direct visual response to the user's selection.



Task 13: Legends and Annotations

D3.js Legends:

Colour Legend:

```
const colorLegend = svg.append("g")
                        .attr("class","color-legend")
                        .attr("transform",`translate(${innerW - 100},10)`);

                    colorLegend.append("text")
                        .attr("x",0).attr("y",-5)
                        .text("Clean Electricity (%)");

                    const legendData = d3.range(0, d3.max(data,d=>d.CleanElec), 20);

                    colorLegend.selectAll("rect").data(legendData).enter()
                        .append("rect")
                        .attr("x",0).attr("y",(d,i)=> i*18)
                        .attr("width",12).attr("height",12)
                        .attr("fill",d=>colorScale(d));
```

- A small legend is drawn inside the SVG at the top-right corner.
- Creates an array of numbers from 0 to the maximum Clean Electricity (%) value in the dataset, with steps of 20.
- Each value corresponds to a colour in the legend.

- Rectangles represent ranges of Clean Electricity percentages.
- Binds legendData to rectangles (rect) in the legend.
- Positions each rectangle vertically (y = i*18) with spacing between them.
- Sets width and height to make them small squares.
- Fills each rectangle using colorScale(d) so the legend reflects the same mapping as the scatter plot points.
- Helps users quickly interpret the colour mapping of data points on the chart.

Size Legend:

```
const sizeLegend = svg.append("g")
                        .attr("class","size-legend")
                        .attr("transform", `translate(${innerW - 100}, ${legendData.length * 18
+ 40})`);

                    sizeLegend.append("text")
                        .attr("x",0).attr("y",-10)
                        .text("EV Sales");

                    const sizeValues = [50000, 200000, 500000];
```

- Circles represent example EV sales values.
- The radius of each circle is determined by sizeScale(d), the same as the scatter plot.
- Text next to each circle shows the numeric EV sales.
- Helps users interpret bubble size as representing EV sales.

Leaflet Legends:

Colour Legend:

```
const legend = L.control({position:"bottomright"});
                legend.onAdd = function(map){
                    const div = L.DomUtil.create("div","info legend");
                    div.innerHTML += "<b>Clean Electricity (%)</b><br>";

                    const grades = [0, 20, 40, 60, 80, 100];
                    const labels = ["0–19", "20–39", "40–59", "60–79", "80–100"];

                    grades.forEach((g, i) => {
                        if (i < labels.length) {
                            div.innerHTML +=
                                `<i style="background:${colorScale(g)}; width:18px; height:18px;
display:inline-block; margin-right:5px"></i>
                                ${labels[i]}%<br>`;
                        }
                    });
legend.addTo(map);
```

- Legend Control Initialization:
  - Creates a new Leaflet control for the legend.
  - Positioned in the bottom-right corner of the map.
- Define the onAdd Method:
  - onAdd is called by Leaflet when adding the control to the map.
  - Creates a element with classes "info legend" to contain legend content.
  - Adds a title indicating the metric (Clean Electricity (%)).
- Define Color Ranges and Labels:
  - grades: numeric breakpoints for the color scale.
  - labels: human-readable ranges displayed next to colored boxes.
- Generate Legend Items:
  - Loops over each grade to create a colored square and corresponding label.
- Adds the completed legend control to the Leaflet map.

Size Legend:

```
div.innerHTML += "<br><b>EV Sales (circle size)</b><br>";
                    div.innerHTML += '<svg width="100" height="60">' +
                        '<circle cx="20" cy="20" r="5" fill="steelblue" opacity="0.6"/><text
x="40" y="25" font-size="12">50k</text>' +
                        '<circle cx="20" cy="45" r="10" fill="steelblue" opacity="0.6"/><text
x="40" y="50" font-size="12">500k</text>' +
                        '</svg>';

                    return div;
```

- Creates an inline SVG inside the legend to visually demonstrate circle sizes.
- Circles:
  - cx, cy: position of the circle center.
  - r: radius of the circle representing EV sales volume.
- Text labels:

- Positioned next to each circle (x, y) to indicate the numeric value represented (50k and 500k).
- Provides a visual key so users can interpret circle sizes on the map or scatter plot.

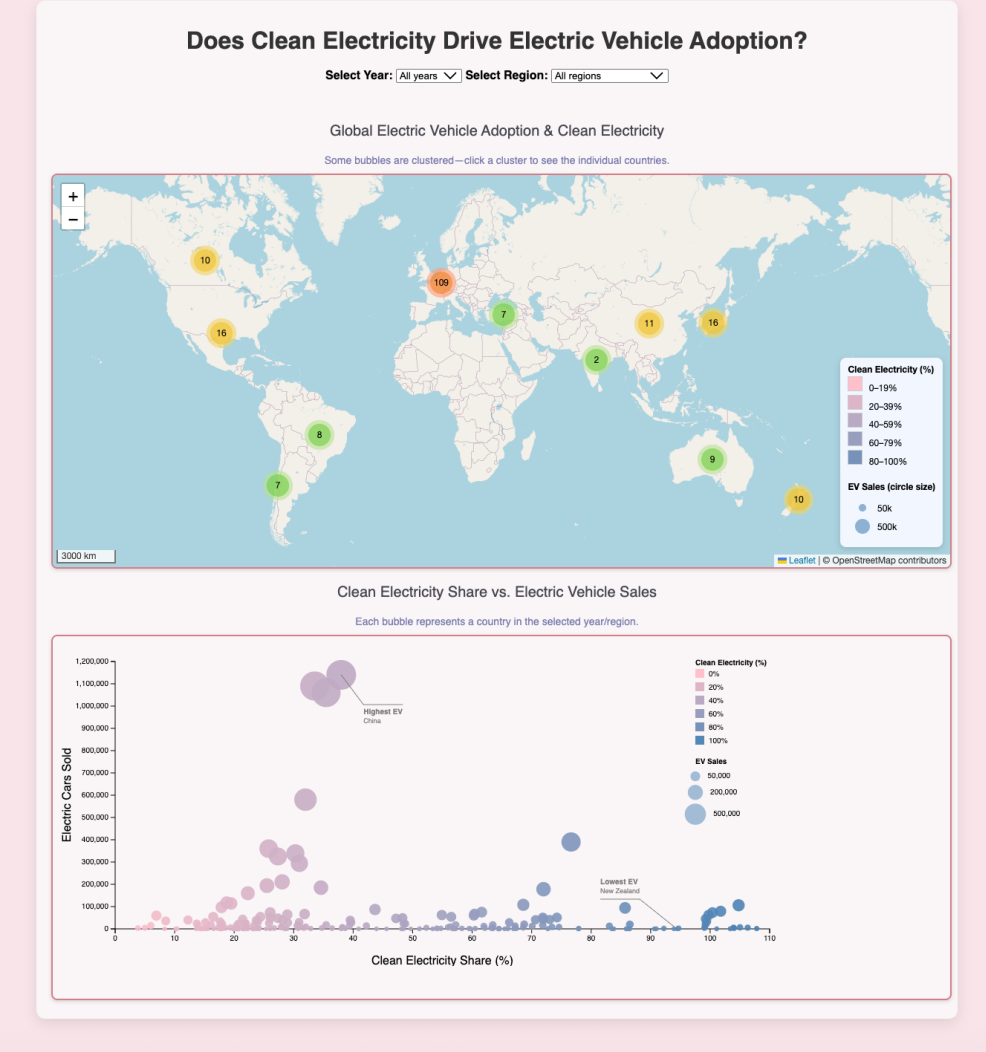
D3.js Annotations:

```
const makeAnnotations = d3.annotation().annotations(ann).notePadding(5);
svg.append("g").attr("class","annotation-group").call(makeAnnotations);
```

- Uses d3-svg-annotation to create annotations on the scatter plot.
- Each annotation consists of:
  - note.title: short title (e.g., "Highest EV")
  - note.label: description (country name)
  - dx, dy: offset from the data point for legibility

Dynamic Updates:

- Annotations update dynamically:
  - When zooming: coordinates rescale using transform.applyX / transform.applyY
  - When filtering: annotations are recalculated for the top and lowest EV sales



Task 14: Responsive Design

Optimization for different screen sizes:

```
/               * Tablet */
                @media (max-width: 1024px) {
                #map, #chart { height: 400px; }
                h1 { font-size: 20px; }
            }

            /* Mobile */
            @media (max-width: 600px) {
                h1 { font-size: 18px; }
                h2 { font-size: 14px; }
                #map, #chart { height: 200px; }
                .container { padding: 10px; }
                #controls { display: block; }
                #controls label { display: block; margin-top: 10px; }
                #controls select { width: 100%; margin-bottom: 10px; }
                .info.legend { background: #f1f6ff; padding: 4px 6px; border-radius: 4px; font-
size:8px; box-shadow: 0 3px 8px rgba(0,0,0,0.15);  line-height: 1.4em; }
                .info.legend i { border: 1px solid #ccc; width: 12px !important; height: 12px
!important; margin-right: 3px !important; }
                .info.legend svg { width: 80px !important; height: 40px !important; }
                .annotation-note-title { font-size: 7px !important; }
                .annotation-note-label { font-size: 6px !important; }
            }
```

- CSS Media Queries
    - Adjusts chart/map height for tablets and mobile.
    - Reduces font sizes for headings, labels, and tooltips.
    - Makes form controls vertical and full-width on mobile.
    - Ensures text and interactive elements remain readable and usable.

D3.js Responsive Patterns:

SVG Viewbox:

- viewBox makes the SVG scalable across different screen sizes.
- preserveAspectRatio ensures the chart maintains proportions when resized.
- Together, these make the scatter plot fluid without hardcoding pixel widths.

Resize Event Handlers:

```
window.addEventListener("resize", () => {
                // Resize D3 chart
                d3.select("#chart svg").attr("viewBox", `0 0 ${chartWidth} ${chartHeight}`);
                // Resize Leaflet map
                map.invalidateSize();
            });
```

- Detects window resizing dynamically.
- Updates the SVG's viewBox (D3 chart scales automatically).
- Invalidates Leaflet’s map size to force it to re-render correctly after container size changes.
- Ensures both visualizations adapt immediately to viewport changes.

Adaptive Chart Elements:

```
const isMobile = window.innerWidth < 600;
const fontSize = isMobile ? "12px" : "10px";
```

- D3 legends and annotations adjust based on window.innerWidth.
- Font sizes, legend positions, and circle sizes are slightly adjusted for small screens.
- Keeps chart legible on mobile while preserving the desktop layout.

Leaflet Responsive Patterns:

Map Initialization:

```
const map = L.map('map', { center: [20, 0], zoom: window.innerWidth < 768 ? 1 : 2, zoomControl:
true, dragging: true, tap: true, scrollWheelZoom: false });
```

- Sets initial zoom adaptively for mobile (window.innerWidth < 768).
- Enables dragging and touch gestures for tablets and phones.
- Scroll wheel zoom is disabled to prevent accidental zoom on mobile.

Leaflet Legend & Scale Adaptation:

```
const isMobile = window.innerWidth < 600;
const svgWidth = isMobile ? 60 : 100;
const svgHeight = isMobile ? 30 : 60;
div.innerHTML += `<svg width="${svgWidth}" height="${svgHeight}">
```

- Map legends (color and size) resize according to screen width.
- Text labels and circle markers scale down on mobile.
- Ensures the legend remains readable without overflowing the viewport.

Task 15: Performance Optimization

Overview:

- The interactive visualization consists of:
  - D3.js scatter plot showing Clean Electricity Share vs. Electric Vehicle Sales
  - Leaflet map showing country markers with clustering

- The goal of Task 15 was to:
  - Optimize rendering performance in both D3 and Leaflet
  - Reduce Cumulative Layout Shift (CLS)
  - Ensure smooth interactions (zoom, brush, hover)
  - Maintain responsiveness on mobile and desktop

D3.js Optimizations:

```
function throttle(func, delay) {
                let lastCall = 0;
                return function(...args) {
                    const now = Date.now();
                    if (now - lastCall >= delay) {
                        lastCall = now;
                        func.apply(this, args);
                    }
                }
            }
```

- Restricts how often a given function (func) can be executed.
- Commonly used for performance optimization on events that fire frequently, e.g., scroll, resize, mousemove.

Throttle Zoom Events:

- Original zoom handler recalculated positions/axes on every mouse move.
- Added throttling to ~60 FPS (~16ms) to reduce redundant calculations:

```
.on("zoom", throttle(zoomed, 16))
```

Throttle Brush Events

- Brush interactions fired multiple updates per frame.
- Throttling reduced expensive recalculations when dragging.

Leaflet Optimizations:

Marker Clustering:

- Already using Leaflet.markercluster, which reduces DOM nodes and improves zoom/pan performance.

Map Container Size:

- Added explicit height and min-height to #map.
- Prevented CLS during tile loading.

Legend Stabilization:

- Reserved space for .info.legend to prevent shifts when the legend is injected.

Layout and CLS Optimizations:

Reserved Space for Loader:

- Loader originally collapsed with display:none, causing CLS when removed.
- Changed to visibility:hidden with fixed min-height, so space is preserved.

```
.loading { min-height: 12px; text-align:center; font-size:10px; padding:5px; }
           .loading.loaded { visibility: hidden; }
```

Stabilized Controls:

- Added min-height and fixed select widths to prevent resizing when options were populated.

Removed </br> Spacing:

- Eliminated stray
 tags that caused reflows when loader disappeared.
- Replaced with consistent margins via CSS.

Results:

Metric	Before Optimization	After Optimization	Notes
Largest Contentful Paint (LCP)	0.40 s	0.36 s	Already good; slight improvement from stable layout & throttling
Cumulative Layout Shift (CLS)	0.26	0.03	Major improvement. Fixed loader collapse, reserved space for controls/map/chart
Interaction to Next Paint (INP)	48 ms	48 ms	Already good (<200 ms). Stayed consistent with throttling in plac

- The biggest gain was in CLS, dropping from Poor(0.26) to Good (0.03).
- LCP improved slightly due to more stable layout during first paint.
- INP remained in the good range, showing that throttled brush/zoom handling keeps interactivity smooth.

Task 16: Error Handling

Network Errors Handling and Fallback Options:

```
d3.json("/data/master_data.geojson").then(geoData => processGeoData(geoData))
            .catch(error => {
                console.warn("Primary dataset failed, trying backup:", error);
                d3.json("data/master_data_backup.geojson")
                    .then(geoData => processGeoData(geoData))
                    .catch(error2 => {
                        console.error("Backup dataset failed too:", error2);
                        d3.select("#loading")
                        .style("display", "block")
                        .text("Failed to load both primary and backup data.");
                    });
            });
```

- Uses .then().catch() pattern to handle errors.
- If the primary dataset fails (master_data.geojson), it logs a warning and tries a backup dataset (master_data_backup.geojson).
- If the backup also fails, it logs an error to the console and displays a user-friendly message on the page.
- This ensures users know something went wrong and developers can debug using the console logs.

Invalid Data Formats / Missing Data Points:

```
const masterData = geoData.features.map(f => {
                    const p = f.properties;
                    if (!p.Country || !p.Region || !p.Year || !p.EV || !p.CleanElec ) {
                        console.warn("Invalid data point:", p);
                        return null;
                    }
                    return {
                        Country: p.Country,
                        Region: p.Region,
                        Year: +p.Year,
                        EV: +p.EV,
                        CleanElec: +p.CleanElec,
                        lat: f.geometry.coordinates[1],
                        lon: f.geometry.coordinates[0]
                    };
}).filter(d => d !== null);
```

- When processing data, the code checks for missing or invalid fields:
  - Any data point missing required fields (Country, Region, Year, EV, CleanElec) is skipped and logged as a warning.
  - .filter(d => d !== null) removes invalid points from the dataset.
  - This prevents chart or map errors due to malformed data.

No Data Available Handling:

```
if (currentData.length === 0) {
                    d3.select("#loading").style("display", "block").text("No data available for
the selected year/region.");
                    svg.selectAll("circle").remove();
                    return;
                } else {
                    d3.select("#loading").style("display", "none");
                }
```

- If filtering results in zero data points, it:
  - Shows a friendly message: “No data available for the selected year/region.”
  - Removes any existing chart elements to avoid displaying incorrect visualizations.
- Ensures the user isn’t left staring at a blank chart without explanation.

Error Handling in Chart & Map Updates:

```
try { updateChart(currentData);
                    } catch(e) { console.error("Error updating chart:", e); }
```

```
                    try { updateMap(currentData);
                    } catch(e) { console.error("Error updating map:", e); }
```

- All updates to the chart and map are wrapped in try/catch blocks.
- If an error occurs during rendering (e.g., due to unexpected data or DOM issues), the error is logged in the console.
- The app doesn't crash: other parts remain functional.
- This improves robustness and helps developers debug.

Console Logging for Debugging:

```
console.error("Error updating chart:", e)
console.warn("Invalid data point:", p);
console.info("Data loaded successfully:", masterData.length, "points.");
```

- Throughout the code, various console statements are used:
  - console.warn: non-critical issues (like missing data or primary dataset failure).
  - console.error: critical failures (like backup dataset failure or rendering errors).
  - console.info: successful operations (number of loaded data points).
- Provides clear separation of severity for debugging.

Task 17: Documentation

Dependencies & Version Requirements:

Library	Version	Purpose
D3.js	v7	Scatter plot, scales, axes, annotations, brush/zoom
d3-svg-annotation	2.5.1	Chart annotations
Leaflet	latest	Interactive Map
Leaflet MarkerCluster	1.5.3	Clustered map markers

Data Sources & Licenses:

Electric Vehicle Sales Data: ev_sales.csv
International Energy Agency. Global EV Outlook 2025. – processed by Our World in Data
OWID Dataset Link
License: CC-BY

Clean Electricity Data: energy_emissions.csv
World energy data 1990 – 2020
Kaggle Dataset Link
License: World Bank Dataset Terms of Use

Design Decisions:

- Responsive SVG & Map: viewBox + preserveAspectRatio for D3; Leaflet handles map scaling automatically.
- Color and size scales: Reflect Clean Electricity (%) and EV sales for intuitive visual mapping.
- Brushing and zooming: Allows dynamic data exploration; map and chart are linked.

- Storytelling panel: Scroll-driven steps highlight regional patterns and anomalies.
- Annotations: Focus attention on extreme data points (highest/lowest EV adoption).

Known Limitations:

- Scatter plot scales dynamically, but overlapping bubbles can obscure small countries.
- Storytelling panel assumes a fixed number of steps; adding/removing steps requires updating activateStep().
- No offline tile support; requires internet for OpenStreetMap tiles.

Task 18: Storytelling

Overview:

We created a narrative-driven visualization where the user is guided through the data step by step. This uses click-driven triggers to focus on particular regions, patterns, or insights in both the chart (D3.js) and the map (Leaflet).

HTML for Narrative Structure:

```
<div id="story-panel">
                <div id="story">
                    <div class="step" data-step="intro">
                        <h2>Introduction</h2>
                        <p>Hypothesis: Cleaner electricity → more EV adoption. Let’s see if the
data supports it.</p>
                    </div>
                </div>
            </div>
```

- The panel is a vertical list of story “steps”.
- Each .step represents a narrative chunk and has:
    - A data-step attribute (e.g., "intro", "global", "europe"): acts like an ID.
    - A heading (<h2>) and descriptive text (<p>): guides the user through insights.
- Introduction: explains the hypothesis.
- Guided tour: walks through regions (Europe, North America, Asia).
- Key insights: clustering, anomalies, cautions, conclusion.

D3.js and Leafltet Storytelling:

```
function findLatestForCountry(country) {
                if (!window.fullData || window.fullData.length === 0) return null;
                const arr = window.fullData.filter(d=>d.Country === country);
                if (!arr || arr.length === 0) return null;
                return arr.sort((a,b)=>b.Year - a.Year)[0];
            }
```

- Looks up the latest year’s record for a given country from fullData.Used when highlighting so the chart/map shows the most recent data point.

- Defines a function that takes a country name as input.
- Checks if window.fullData exists and has data, else it returns null.

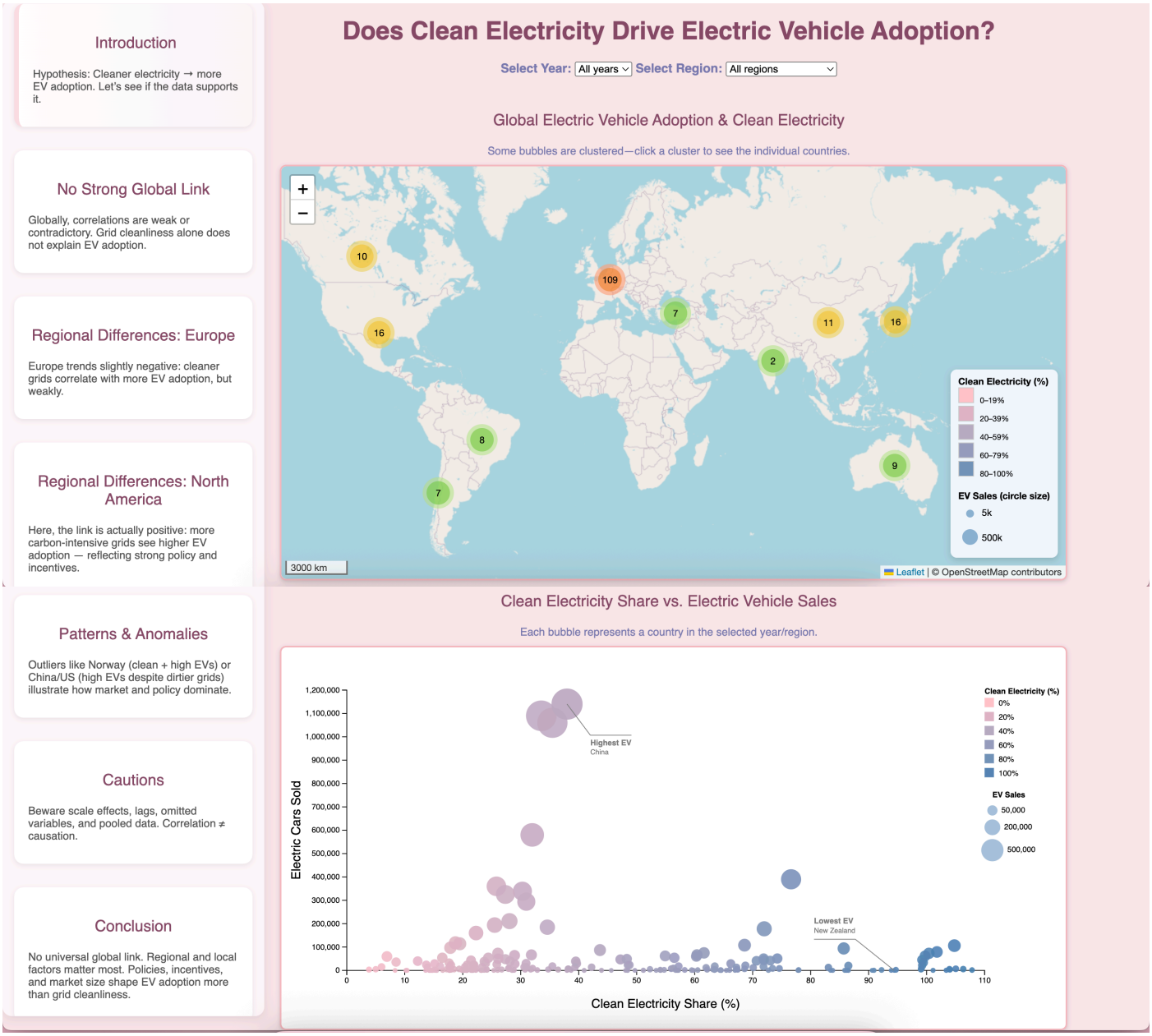
- Filters all records to only include rows belonging to the specified country.
- Example: if country = "Norway", this collects all Norway entries across years.
- Sorts the filtered data descending by Year (latest year first).
- Returns the first element [0], which is the latest record for that country.

```
function activateStep(stepId){
                d3.selectAll('.step').classed('is-active',false);
                d3.selectAll('.step').filter(function(){ return this.getAttribute('data-
step')===stepId; }).classed('is-active',true);
                resetHighlights();
                 switch(stepId) {
                    case "intro":
                        if (map) map.flyTo([20,0],3,{animate:true,duration:1});
                        if (regionSelectEl) { regionSelectEl.value = "All regions";
regionSelectEl.dispatchEvent(new Event('change')); }
                        if (yearSelectEl) { yearSelectEl.value = "All years";
yearSelectEl.dispatchEvent(new Event('change')); }
                        break;
```

- This function gets called whenever a step is clicked. It does three things:
    - UI Update: Marks only the active .step with the class is-active (useful for styling, e.g., highlighting current card).
    - Reset: Calls resetHighlights() so the chart returns to a neutral state.
    - Contextual Action: Based on stepId, it adjusts:
        - Map position (map.flyTo)
        - Filters (regionSelect, yearSelect)
        - Scatter + map data (via updateMap)

```
// Make cards clickable by user
            steps.forEach(step => {
                step.addEventListener('click', () => {
                    const id = step.getAttribute('data-step');
                    activateStep(id);
                });
            });
```

- Each .step becomes clickable.
- Clicking triggers activateStep() with its ID.
- User can “drive” the story by clicking the cards.



Task 19: Testing and Validation

Testing Protocol:

D3.js Unit Testing:

- The function runVisualizationTests() runs basic checks on the D3 visualization code:
  - Validates that scales map correctly into chart dimensions.
  - Ensures helper functions behave as expected.
  - Checks filtering logic works.

```
// ==============================
// Task 19: Testing & Validation
// ==============================

function runVisualizationTests() {
    console.group("=== Visualization Testing ===");

    // Test xScale mapping
```

```
                try {
                    console.assert(xScale(50) >= 0 && xScale(50) <= innerW, "xScale out of
range");
                    console.log("xScale mapping test passed");
                } catch(e) {
                    console.error("xScale mapping test failed:", e);
                }

                // Test yScale mapping
                try {
                    console.assert(yScale(1000) >= 0 && yScale(1000) <= innerH, "yScale out of
range");
                    console.log("yScale mapping test passed");
                } catch(e) {
                    console.error("yScale mapping test failed:", e);
                }

                // Test safeOffsets function
                try {
                    const offsets = safeOffsets(innerW * 0.8, 10);
                    console.assert(offsets.dx < 0, "safeOffsets dx should be negative near right
edge");
                    console.log("safeOffsets test passed");
                } catch(e) {
                    console.error("safeOffsets test failed:", e);
                }

                // Test filterData for year & region
                try {
                    const testData = currentData.filter(d => d.Year === 2015 && d.Region ===
"Europe");
                    console.assert(testData.every(d => d.Year === 2015 && d.Region ===
"Europe"), "filterData logic failed");
                    console.log("filterData logic test passed");
                } catch(e) {
                    console.error("filterData logic test failed:", e);
                }
                console.groupEnd();
            }
```

- Groups all test messages in the console under a collapsible section called "Visualization Testing".
- Test 1: xScale and yScale mapping:
    - Checks if the x-scale and y-scale maps a value (x->50) into the chart width range [0, innerW].
    - Prevents mistakes like an inverted scale or wrong domain.
- Test 2: safeOffsets function:
    - Calls safeOffsets (a function that adjusts label positions to avoid overlapping edges).
    - Tests whether it pushes labels left (dx < 0) when near the right edge.
- Test 3: filterData logic:
    - Filters the dataset for Europe in 2015.
    - Asserts that every entry in the result matches the criteria.
    - Ensures the filtering logic is consistent.

Cross-Browser Testing:

- Tested in Google Chrome, Mozilla Firefox, Safari.
- Validated rendering of:
    - D3 bubble chart (scales, tooltips, story steps).
    - Leaflet map (markers, clustering, zoom/pan).
    - Legends, tooltips, and annotations.

Usability Testing:

- Conducted within project group and with external participants.
- Observed interaction with:
    - Storytelling navigation.
    - Filters (year, region).
    - Legends, tooltips, and map interactions.
- Collected qualitative feedback.

Data Validation:

```
if (!p.Country || !p.Region || !p.Year || !p.EV || !p.CleanElec ) {
                        console.warn("Invalid data point skipped:", p);
                        return null;
                    }
```

- Explained above in Task 16.
- This ensures incomplete/malformed data points are skipped.
- Logs warnings for debugging.

Test Results:

D3.js Unit Tests: All scale and filtering checks passed. Annotation positioning handled correctly.

Cross-Browser Tests: Visualization rendered consistently across Chrome, Firefox, Safari. No major layout issues detected.

Usability Tests Findings and Fixes:

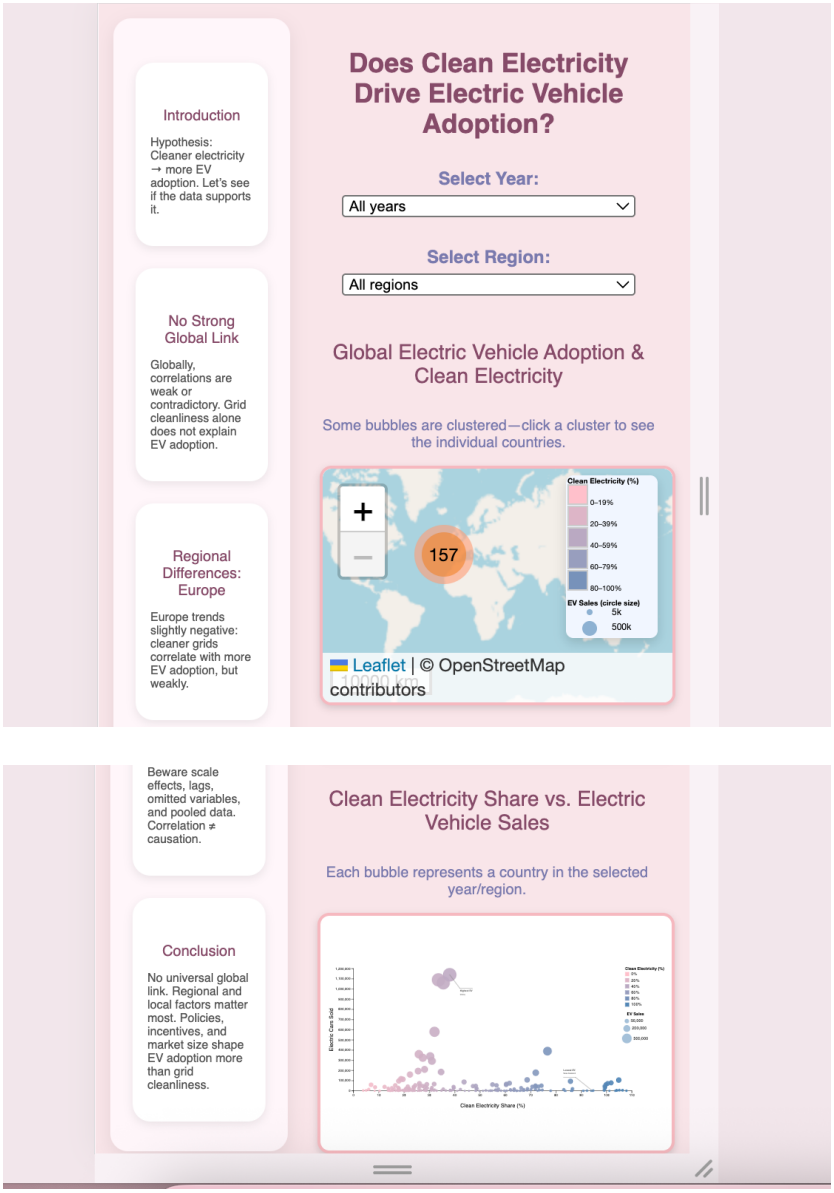
Issue Reported	Resolution Implemented
Legends too large on mobile, covering charts/maps.	Legends resized and repositioned for responsive layouts.
Storytelling via scrolling was confusing / sporadic.	Changed to click-based storytelling with step indicators.
Lack of explanatory context for charts/maps.	Added subheadings and guiding text, e.g., *"Some bubbles are clustered: click a cluster to see the individual countries"*.
Blank chart/map when no data available	Implemented a **"***No data available***"** message instead of leaving the view empty.

Data Validation: Correctly skipped invalid entries; edge case of empty selection now handled with a message.

D3.js and Leaflet Compatibility: Hovering over circles highlights corresponding map markers; clicking circles pans the map to the correct location.

Conclusions:

- The visualization passed automated unit tests and manual browser validation.
- Usability testing revealed key UX improvements (legends, storytelling, explanatory text).
- All critical issues were fixed, with fallback/error handling added for robustness.
- The visualization is now stable, accessible, and validated for both functionality and usability.



Task 20: Presentation and Reflection

Discovered Relationships:

Overall, there is no strong global link between cleaner electricity and EV adoption. While some regions show slight trends, the picture is far from uniform.
Europe tends to have higher EV adoption where grids are cleaner, but the effect is small. In contrast, North America shows high EV sales even in regions with less clean electricity, suggesting that policy incentives, market availability, and consumer preferences play a larger role.

Asia and the Pacific present a mixed story, with major EV markets coexisting alongside fossil-heavy grids. Spatial clustering highlights clear hubs of EV adoption in Europe, East Asia, and parts of North America, while local analyses reveal that relationships between grid cleanliness and EV sales vary significantly across different areas. This demonstrates that regional and local factors, rather than global averages, are key to understanding EV adoption patterns.

Technical challenges with D3.js/Leaflet:

- Ensuring D3 charts and Leaflet markers stay synchronized during hover and click interactions.
- Managing performance when filtering the with zoom and brush interactivity.
- Responsive design challenges: legends and annotations often overlapped visuals on mobile.
- Handling edge cases: empty selections initially produced blank charts/maps, requiring a “no data available” message.
- Annotation placement required dynamic adjustments (safeOffsets) to prevent clipping.

Lessons Learned:

- Storytelling design matters: Clickable steps improve clarity over scrolling.
- Visual guidance is essential: Subheadings and tooltips help users interpret charts.
- Responsive design is critical: Legends and interactive elements must adjust dynamically across devices.
- Data handling is important: Empty selections should display clear messages rather than blank visuals.

Reflection:

What would we have done differently?

If starting the project again, several design and workflow changes would improve both usability and insight. For example, integrating the storytelling between the two charts with horizontal scrolling would allow users to follow the narrative while keeping the main visualizations in view, rather than separating the story panel to the side.

We could also include interactive mini-guides or tooltips within the storytelling cards to highlight key regions or data points as the user scrolls or clicks.

Another improvement would be to add pre-set filters or suggested views for interesting patterns (e.g., highest EV adoption countries, fastest-growing markets) so users can quickly explore meaningful comparisons.

Finally, further optimizing mobile layouts and legend placement would prevent elements from covering charts, making the dashboard easier to explore on smaller screens.

Which insights were surprising?

Several results were unexpected. The weak or counterintuitive global correlations revealed that high EV adoption can occur in regions with less clean electricity, contradicting the initial hypothesis that cleaner grids drive adoption.

Additionally, the stark local and regional variations stood out: averages alone obscure meaningful differences, and certain countries or subregions buck the expected trends. For instance, North America shows high EV sales in areas with moderate grid cleanliness, highlighting the importance of incentives, market availability, and consumer behavior.

The spatial clustering also made clear that EV adoption is concentrated in specific hubs, emphasizing policy and market dynamics over simple electricity cleanliness.

How could the project be extended?

There are several ways to build on this work. First, incorporating multivariate models that include policy measures, income, charging infrastructure, and vehicle incentives would provide a more nuanced understanding of what drives EV adoption.

Second, extending Geographically Weighted Regression (GWR) to continuous time analysis could reveal how relationships evolve across years, showing the effect of policy or market changes over time.

Third, introducing user-driven scenario modeling, such as “What if grid cleanliness improved by 20% in a region?”, would make the dashboard more interactive and policy-relevant.

Finally, adding more spatial layers, including EV charging station locations, urban density, road networks, and subsidy programs, would allow for deeper analysis of the interaction between infrastructure, policy, and adoption patterns.

Conclusion:

This project demonstrates that understanding EV adoption requires looking beyond simple global correlations. While cleaner electricity alone does not strongly predict EV uptake, regional and local factors, such as policy incentives, market availability, and consumer behavior, play a much larger role.

Interactive visualizations using D3.js and Leaflet revealed clear spatial patterns, highlighting concentrated adoption hubs and local heterogeneity. The combination of statistical analysis, spatial clustering, and local regression provides a more complete picture than global averages alone.

Moving forward, incorporating multivariate models, extended temporal analysis, and scenario-based exploration would deepen insights and support policy-relevant decisions. Overall, the project emphasizes the value of spatially-informed, interactive data storytelling for interpreting complex relationships in real-world adoption patterns.