<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<title>UWBe Impact Atlas – Climate Action : One Billion Drops</title>

<meta name="viewport" content="width=device-width, initial-scale=1" />

<!-- Mapbox GL JS -->

<link href="https://api.mapbox.com/mapbox-gl-js/v3.12.0/mapbox-gl.css" rel="stylesheet"/>

<script src="https://api.mapbox.com/mapbox-gl-js/v3.12.0/mapbox-gl.js"></script>

<!-- Turf.js -->

<script src="https://cdn.jsdelivr.net/npm/@turf/turf@6/turf.min.js"></script>

<!-- PapaParse -->

<script src="https://cdn.jsdelivr.net/npm/papaparse@5.4.1/papaparse.min.js"></script>

<style>

:root { --header-h: 64px; }

html, body { margin: 0; padding: 0; height: 100%; overflow: hidden; }

.header-panel { position: fixed; top: 0; left: 0; right: 0; z-index: 1000;

display: flex; justify-content: space-between; align-items: center;

background: #fff; padding: 10px 30px; border-bottom: 2px solid #eee;

box-shadow: 0 2px 8px rgba(0,0,0,.06);

font-family: 'Georgia', serif; height: var(--header-h); font-size: 16px;

}

.header-panel .logo img { height: 52px; }

.header-panel .title { text-align: center; flex: 1; }

.header-panel .title h1 { margin: 0; font-size: 22px; color: #1a3dab; }

.header-panel .title h2 { margin: 0; font-size: 18px; color: #e63323; }

.header-panel .menu a { text-decoration: none; color: #444; font-weight: 700; font-size: 16px; }

#map {

position: fixed;

top: var(--header-h);

left: 0;

right: 0;

bottom: 0; /\* fills remaining viewport \*/

}

/\* Inset: hydrological context (bottom-right) \*/

#inset{

position:absolute; right:12px; bottom:12px;

width:min(32vw,560px); height:min(32vh,420px);

z-index:902; background:#fff; border:1px solid #d1d5db;

border-radius:12px; overflow:hidden; box-shadow:0 1px 5px rgba(0,0,0,.28);

}

/\* Filters card (top-left) \*/

#controls { position: absolute; top: calc(var(--header-h) + 10px); left: 12px;

width: 300px; z-index: 900; background: #fff; border-radius: 12px;

box-shadow: 0 2px 10px rgba(0,0,0,.15); padding: 14px;

font-family: system-ui, -apple-system, Segoe UI, Roboto, Arial, sans-serif; font-size: 14px;

}

#controls label { display:block; font-weight:700; margin: 6px 0 4px; font-size: 14px; }

#controls select { width: 100%; padding: 8px 10px; border: 1px solid #d1d5db; border-radius: 10px; font-size: 14px; }

.muted { color:#6b7280; font-size:12px; }

.row { display: grid; gap: 8px; margin-top: 8px; }

.error { background:#fee2e2; border:1px solid #fca5a5; color:#7f1d1d; padding:10px; border-radius:8px; font-size:13px; display:none; margin-top:8px; }

/\* Reusable card + bold numbers (summary + legend) \*/

.card {

background:#fff;

padding:10px 12px;

border-radius:10px;

box-shadow:0 1px 4px rgba(0,0,0,.25);

font-size:13px;

line-height:1.4;

}

.card strong, .card .num { font-weight:700; }

/\* Summary card sits inside controls, under the dropdowns \*/

#summaryCard { margin-top: 10px; }

/\* Legend: bottom-left, fixed width with inner scroll for donors \*/

#legend{

position:absolute; left:12px; bottom:12px;

width: 300px; /\* tidy, fixed width \*/

max-height: 38vh; /\* overall max height \*/

overflow: hidden; /\* outer stays neat; inner scroll handles overflow \*/

z-index:903;

}

.legend-item{ display:flex; align-items:center; gap:8px; margin:4px 0; }

.legend-line{ height:0; border-top:3px solid; width:30px; }

.legend-dot{ width:14px; height:14px; border-radius:50%; display:inline-block; border:1px solid #fff; box-shadow:0 0 1px rgba(0,0,0,.6); }

.legend-title{ font-weight:700; margin:2px 0 6px 0; font-size:13px; }

.legend-scroll{

margin-top:8px; padding-top:8px;

border-top:1px solid #e5e7eb;

max-height: 28vh; /\* scroll area height \*/

overflow-y: auto;

}

</style>

</head>

<body>

<div class="header-panel">

<div class="logo"><img src="Logo.png" alt="United Way Logo" /></div>

<div class="title"><h1>UWBe Impact Atlas</h1><h2>Climate Action : One Billion Drops</h2></div>

<div class="menu"><a href="#" onclick="return false;">ABOUT</a></div>

</div>

<div id="controls">

<label for="donorSel">Select Donor</label>

<select id="donorSel"><option value="">All Donors…</option></select>

<label for="fySel">Select Financial Year</label>

<select id="fySel"><option value="">All Years…</option></select>

<div class="row">

<label for="zoneSel">Zone</label>

<select id="zoneSel" disabled><option value="">Select donor/FY first…</option></select>

</div>

<div class="muted">Tip: choose a Donor or FY to reveal relevant Zones.</div>

<!-- SUMMARY: uses same .card look and bold numbers -->

<div id="summaryCard" class="card counts">

<div id="countLoc">Locations : <span class="num">–</span></div>

<div id="countWells">Wells : <span class="num">–</span></div>

<div id="narrative"></div>

</div>

<div id="err" class="error"></div>

</div>

<div id="map"></div>

<!-- LEGEND: bottom-left card with scrollable donors -->

<div id="legend" class="card" aria-live="polite"></div>

<div id="inset" aria-hidden="true"></div>

<script>

mapboxgl.accessToken = 'pk.eyJ1IjoicGFua2FqdXdiIiwiYSI6ImNtY3lxdzFwaDByMWIycnB1Z3FqNWZkZHQifQ.ZTvhU5DjLI5x4OwkmXU14w';

/\* ---------- Files ---------- \*/

const FILES = {

district : 'District\_Boundary\_Final\_CRS.geojson',

zones : 'BBMP\_Zones\_Final\_CRS.geojson',

wards : 'BBMP\_Wards\_225\_Final\_CRS.geojson',

basin : 'Bengaluru\_Urban\_Hydrosheds.geojson', // valleys/basins

subws : 'Sub\_Watershed\_Names\_Final\_CRS.geojson', // sub-watersheds (labels from ws\_Name)

microws : 'BBMP\_Micro\_Watershed\_Final\_CRS.geojson' , // micro-watersheds

wells\_pts: 'wells.csv'

};

/\* ---------- State ---------- \*/

const store = { geo:{}, bbmp:{}, ui:{ zonesShown:false } };

let map, inset;

let wellsAll = {type:'FeatureCollection',features:[]};

let wellsFiltered = {type:'FeatureCollection',features:[]};

let wellPointsAll = {type:'FeatureCollection',features:[]};

const wellsByLoc = new Map(); // \_locNorm -> Feature[]

const bboxByLoc = new Map(); // \_locNorm -> [minX,minY,maxX,maxY]

let donorStats = new Map(); // donor -> {color,totalWells,totalLocations,years:Set}

/\* ---------- Helpers ---------- \*/

const errEl=()=>document.getElementById('err');

function showError(m){const e=errEl(); e.textContent=m; e.style.display='block'; console.error(m);}

function clearError(){const e=errEl(); e.textContent=''; e.style.display='none';}

const plural=(n,one,more)=>n===1?one:more;

const norm=(str)=>String(str||'').trim().toLowerCase().replace(/\s+/g,' ');

const ZONE\_AUTO\_ZOOM = 9.8; // trigger zoom level

/\* Canonical FY from “Year of Completion” \*/

function canonicalFY(raw){

if (raw==null) return '';

let s = String(raw).trim();

if (!s) return '';

// Already "YYYY-YYYY" or "YYYY-YY"

let m = s.match(/^(\d{4})\s\*[-–\/]\s\*(\d{2,4})$/);

if (m){

const start = +m[1];

let end = +m[2]; if (end<100) end = 2000+end;

return `${start}-${end}`;

}

// "Mon-YY" / "Mon YYYY"

const MON = {jan:1,feb:2,mar:3,apr:4,may:5,jun:6,jul:7,aug:8,sep:9,oct:10,nov:11,dec:12};

m = s.match(/^([A-Za-z]{3,})[\s\-\/]?(\d{2,4})$/);

if (m){

const mon = (MON[m[1].slice(0,3).toLowerCase()] || 3);

let year = +m[2]; if (year<100) year = 2000+year;

const start = (mon>=4) ? year : (year-1);

return `${start}-${start+1}`;

}

// "03/2019" or "2019-03"

m = s.match(/^(\d{1,2})[\/\-](\d{2,4})$/) || s.match(/^(\d{4})[\/\-](\d{1,2})$/);

if (m){

let a = +m[1], b = +m[2]; let mon, year;

if (a>12){ year=a; mon=b; } else { mon=a; year=b; }

if (year<100) year = 2000+year;

const start = (mon>=4) ? year : (year-1);

return `${start}-${start+1}`;

}

// Just a year

m = s.match(/^\d{4}$/);

if (m){ const y=+s; return `${y-1}-${y}`; }

return '';

}

/\* Make a compact range like "2022–2025" from a Set of FY strings "YYYY-YYYY" \*/

function compactFYRange(yearSet){

if (!yearSet || !yearSet.size) return '';

const starts = [...yearSet].map(fy => +fy.split('-')[0]).sort((a,b)=>a-b);

const minS = starts[0], maxS = starts[starts.length-1];

return `${minS}–${maxS+1}`;

}

function getName(level, props){

const keys = {

zones: ['ZoneName','ZONE\_NAME','ZONE','zone\_name','zone','Name','NAME'],

wards: ['KGISWardName','WardName','WARD\_NAME','NAME'],

district: ['district','DISTRICT','NAME']

}[level] || ['NAME'];

for (const k of keys){ if (props && props[k]!=null && String(props[k]).trim()!=='') return String(props[k]); }

return '(unnamed)';

}

async function getGeo(key){

if (store.geo[key]) return store.geo[key];

const res = await fetch(FILES[key]); if(!res.ok) throw new Error(`Failed to load ${FILES[key]}`);

const g = await res.json(); if(!g || !g.features) throw new Error(`Invalid GeoJSON in ${FILES[key]}`);

g.features.forEach((f,i)=>{ if(!f.properties) f.properties={}; f.properties.\_\_uid\_\_=`${key}:${i}`; });

store.geo[key]=g;

if(map && !map.getSource(key)) map.addSource(key,{type:'geojson',data:g});

return g;

}

function populateSelect(levelKey, sel, feats){

const pretty = { zones:'Zones' };

sel.innerHTML = `<option value="">All ${pretty[levelKey]||levelKey}…</option>`;

feats.slice().sort((a,b)=>getName(levelKey,a.properties).localeCompare(getName(levelKey,b.properties)))

.forEach(f=>{

const o=document.createElement('option');

o.value=f.properties.\_\_uid\_\_;

o.textContent=getName(levelKey,f.properties);

sel.appendChild(o);

});

sel.disabled = feats.length===0;

}

function sumWells(features){ return features.reduce((n,f)=> n + (Number(f.properties.wells\_count)||0), 0); }

/\* ---------- Donor colors & stats ---------- \*/

const DONOR\_COLORS = [

'#1f77b4','#ff7f0e','#2ca02c','#d62728','#9467bd','#8c564b','#e377c2','#7f7f7f',

'#bcbd22','#17becf','#e41a1c','#377eb8','#4daf4a','#984ea3','#ff7f00','#ffff33',

'#a65628','#f781bf','#999999','#66c2a5','#fc8d62','#8da0cb','#e78ac3','#a6d854',

'#ffd92f','#e5c494','#b3b3b3'

];

function colorForDonor(d){ // stable hash to color index

let h=0; for(let i=0;i<d.length;i++){ h=(h\*31 + d.charCodeAt(i))|0; }

const idx = Math.abs(h) % DONOR\_COLORS.length;

return DONOR\_COLORS[idx];

}

function buildDonorStats(){

donorStats.clear();

wellsAll.features.forEach(f=>{

const d = f.properties.donor || 'Unknown';

if(!donorStats.has(d)) donorStats.set(d,{ color: colorForDonor(d), totalWells:0, totalLocations:0, years:new Set() });

});

const byDonorLocations = new Map(); // donor -> Set of unique lat/lon keys

wellsAll.features.forEach(f=>{

const d = f.properties.donor || 'Unknown';

const key = `${f.geometry.coordinates.join(',')}`;

if(!byDonorLocations.has(d)) byDonorLocations.set(d,new Set());

byDonorLocations.get(d).add(key);

const s = donorStats.get(d);

s.totalWells += (+f.properties.wells\_count||0);

if (f.properties.fy) s.years.add(f.properties.fy);

});

for(const [d,set] of byDonorLocations.entries()) donorStats.get(d).totalLocations = set.size;

}

/\* LEGEND render (admin + donor list) \*/

function renderLegend(){

const zonesVisible = map && map.getLayer('zones-line') && map.getLayoutProperty('zones-line','visibility')!=='none';

const adminHTML = `

<div class="legend-item"><span class="legend-line" style="border-color:#1255d7"></span> Bengaluru Urban District</div>

<div class="legend-item"><span class="legend-line" style="border-color:#e11d48"></span> BBMP Boundary</div>

${zonesVisible ? '<div class="legend-item"><span class="legend-line" style="border-color:#0e7490"></span> Zones</div>' : ''}

<div class="legend-item"><span class="legend-dot" style="background:#999"></span> Percolation well locations</div>

<div class="legend-title">All donors</div>

`;

const donorsSorted = [...donorStats.entries()].sort((a,b)=>b[1].totalWells - a[1].totalWells);

const donorsHTML = donorsSorted.map(([name,st]) =>

`<div class="legend-item" title="${name}">

<span class="legend-dot" style="background:${st.color}"></span>

<span style="flex:1;white-space:nowrap;overflow:hidden;text-overflow:ellipsis;">${name}</span>

<span class="num">${st.totalWells}</span>

</div>`

).join('');

document.getElementById('legend').innerHTML =

`<div class="card">${adminHTML}</div><div class="legend-scroll">${donorsHTML}</div>`;

}

/\* Update summary card \*/

function updateSummaryCard(){

const donor = document.getElementById('donorSel').value || 'Donors';

const fy = document.getElementById('fySel').value;

const zones = new Set(wellsFiltered.features.map(f=>f.properties.\_zoneName).filter(Boolean));

const totalW = sumWells(wellsFiltered.features);

const totalL = wellsFiltered.features.length;

document.getElementById('countLoc').innerHTML = `Locations : <span class="num">${totalL}</span>`;

document.getElementById('countWells').innerHTML = `Wells : <span class="num">${totalW}</span>`;

let msg = `${donor} supported <span class="num">${totalW}</span> percolation ${plural(totalW,'well','wells')} across <span class="num">${totalL}</span> ${plural(totalL,'location','locations')} in BBMP`;

if (fy) msg += `, FY <span class="num">${fy}</span>`;

if (zones.size) msg += `; Zones: ${[...zones].sort().join(', ')}`;

msg += '.';

document.getElementById('narrative').innerHTML = msg;

}

/\* ---------- Build wells + joins ---------- \*/

async function buildWells(rows){

const cols = rows.length?Object.keys(rows[0]):[];

const donorKey = cols.find(c=>/^donor[ \_]?name$/i.test(c)) || 'donor name';

const locKey = cols.find(c=>/^location[ \_]?name$/i.test(c)) || 'Location Name';

const yocKey = cols.find(c=>/^year[ \_]?of[ \_]?completion$/i.test(c)) || 'Year of Completion';

const wellsKey = cols.find(c=>c.trim().toLowerCase()==='no of percolation well constructed') || 'No of Percolation well Constructed ';

const latKey = cols.find(c=>/latitude|^lat$/i.test(c)) || 'lat';

const lonKey = cols.find(c=>/longitude|lon|lng|^long$/i.test(c)) || 'lon';

wellsAll = {

type:'FeatureCollection',

features: rows

.filter(r => r && r[latKey]!=null && r[lonKey]!=null && String(r[latKey]).trim()!=='' && String(r[lonKey]).trim()!=='')

.map(r => ({

type:'Feature',

geometry:{ type:'Point', coordinates:[ Number(r[lonKey]), Number(r[latKey]) ] },

properties:{

donor: String(r[donorKey] ?? '').trim(),

location: String(r[locKey] ?? '').trim(),

fy: canonicalFY(r[yocKey]),

wells\_count: Number(r[wellsKey]) || 0

}

}))

};

// Spatial attach: zone via ward polygon hit or zone polygon fallback

const wards = store.geo.wards.features;

const zones = store.geo.zones.features;

wellsAll.features.forEach(pt=>{

const hitW = wards.find(w => turf.booleanPointInPolygon(pt, w));

if (hitW){

pt.properties.\_wardName = getName('wards', hitW.properties);

const wardZoneName = hitW.properties.ZoneName || hitW.properties.ZONE\_NAME || hitW.properties.zone || hitW.properties.ZONE || '';

pt.properties.\_zoneName = String(wardZoneName).trim();

const uidByName = store.bbmp.zoneUIDByNormName?.[norm(pt.properties.\_zoneName)];

if (uidByName){

pt.properties.\_zoneUID = uidByName;

} else {

const hitZPoly = zones.find(z => turf.booleanPointInPolygon(pt, z));

if (hitZPoly){

pt.properties.\_zoneUID = hitZPoly.properties.\_\_uid\_\_;

pt.properties.\_zoneName = getName('zones', hitZPoly.properties);

}

}

}

});

wellsFiltered = { type:'FeatureCollection', features: wellsAll.features.slice() };

// Donor stats for legend + popup summary

buildDonorStats();

}

/\* ---------- Zones visibility & filtering on main map ---------- \*/

function toggleAdminVisibility(show){

const v = show ? 'visible' : 'none';

// Keep zones always layout-visible (so they’re hover/clickable); opacity is handled in the layer paint.

if(map.getLayer('bbmp-line')) map.setLayoutProperty('bbmp-line','visibility',v);

renderLegend();

}

function setZonesFilterByUID(zoneUID){

if (!map.getLayer('zones-line')) return;

const hasUID = !!zoneUID;

// Don’t filter the base zone layers; only set the emphasis layer filter

if (map.getLayer('zones-hover')) {

map.setFilter('zones-hover', hasUID ? ['==', ['get','\_\_uid\_\_'], zoneUID] : ['==', ['get','\_\_uid\_\_'], '\_\_none\_\_']);

}

// Base styling stays the same; outlines already fade in by zoom in zones-line

}

function updateZoneVisibility(){

if (!map.getLayer('zones-line')) return;

const zoneSel = document.getElementById('zoneSel');

const zoneUID = zoneSel ? zoneSel.value : '';

// Keep base zones always layout-visible; they fade in by zoom via paint expression

setZonesFilterByUID(zoneUID || '');

renderLegend();

}

/\* ---------- Filters UI ---------- \*/

function setupFiltersUI(){

const donorSel=document.getElementById('donorSel');

const fySel =document.getElementById('fySel');

const zoneSel =document.getElementById('zoneSel');

function setDonorOptions(list, keep){

const val=keep && list.includes(keep)?keep:'';

donorSel.innerHTML = '<option value="">All Donors…</option>' + list.map(d=>`<option${d===val?' selected':''}>${d}</option>`).join('');

}

function setFYOptions(list, keep){

const val=keep && list.includes(keep)?keep:'';

fySel.innerHTML = '<option value="">All Years…</option>' + list.map(y=>`<option${y===val?' selected':''}>${y}</option>`).join('');

}

function rebuildAndFilter(trigger){

const curDonor = donorSel.value;

const curFY = fySel.value;

const curZone = zoneSel.value;

// Rebuild donors limited by FY (if any)

const donorsAll = [...new Set(

wellsAll.features

.filter(f => !curFY || f.properties.fy===curFY)

.map(f => f.properties.donor)

.filter(Boolean)

)].sort();

setDonorOptions(donorsAll, curDonor);

const donorFinal = donorsAll.includes(curDonor) ? curDonor : '';

// Rebuild FY limited by donor (if any)

const fyAll = [...new Set(

wellsAll.features

.filter(f => !donorFinal || f.properties.donor===donorFinal)

.map(f => f.properties.fy)

.filter(Boolean)

)].sort();

setFYOptions(fyAll, curFY);

const fyFinal = fyAll.includes(curFY) ? curFY : '';

// Base set (donor/FY only)

let base = wellsAll.features.filter(f =>

(!donorFinal || f.properties.donor===donorFinal) &&

(!fyFinal || f.properties.fy===fyFinal)

);

// Zones list limited to base set

const zoneUIDs = [...new Set(base.map(f=>f.properties.\_zoneUID).filter(Boolean))];

const zonesSubset = store.geo.zones.features.filter(z => zoneUIDs.includes(z.properties.\_\_uid\_\_));

const prev = (trigger==='zone') ? curZone : '';

populateSelect('zones', zoneSel, zonesSubset);

zoneSel.disabled = zonesSubset.length===0;

if (prev && zoneUIDs.includes(prev)) zoneSel.value = prev;

else if (zonesSubset.length===1) zoneSel.value = zonesSubset[0].properties.\_\_uid\_\_;

const zoneFinal = zoneSel.value;

wellsFiltered.features = base.filter(f => (!zoneFinal || f.properties.\_zoneUID===zoneFinal));

// Update wells layer colors by donor

if (map.getSource('wells')){

map.getSource('wells').setData(wellsFiltered);

const pairs = [];

for (const [d,st] of donorStats.entries()) pairs.push(d, st.color);

map.setPaintProperty('wells-circle','circle-color', ['match',['get','donor'], ...pairs, '#999']);

}

// Show admin only when donor or FY chosen

toggleAdminVisibility(!!donorFinal || !!fyFinal);

// Zoom to filtered points

if (wellsFiltered.features.length){

if (wellsFiltered.features.length===1){

const [x,y]=wellsFiltered.features[0].geometry.coordinates;

const dLon=0.02, dLat=0.02\*Math.cos(y\*Math.PI/180);

map.fitBounds([[x-dLon,y-dLat],[x+dLon,y+dLat]],{padding:70,duration:600,maxZoom:12.6});

} else {

map.fitBounds(turf.bbox(wellsFiltered),{padding:60,duration:700,maxZoom:12.6});

}

}

// Keep inset wells in sync

updateInset();

if (map.getLayer('wellPoints-circle')) {

map.setLayoutProperty('wellPoints-circle','visibility','none');

map.setFilter('wellPoints-circle', ['==', ['get','\_locNorm'], '\_\_none\_\_']);

}

// Update summary + legend + zones visibility

updateSummaryCard();

updateZoneVisibility();

renderLegend();

}

donorSel.addEventListener('change', ()=>rebuildAndFilter('donor'));

fySel .addEventListener('change', ()=>rebuildAndFilter('fy'));

zoneSel.addEventListener('change', ()=>{ rebuildAndFilter('zone'); });

// initial lists

const donors=[...new Set(wellsAll.features.map(f=>f.properties.donor).filter(Boolean))].sort();

const fys =[...new Set(wellsAll.features.map(f=>f.properties.fy).filter(Boolean))].sort();

setDonorOptions(donors, '');

setFYOptions(fys, '');

populateSelect('zones', zoneSel, store.geo.zones.features);

zoneSel.disabled = false;

rebuildAndFilter();

}

/\* ---------- Inset: HYDROLOGY ONLY + wells (no zones here) ---------- \*/

function buildInset(blrFC){

Promise.all([FILES.basin, FILES.subws, FILES.microws].map(u=>fetch(u).then(r=>r.json()))).then(([basin,subws,microws])=>{

inset = new mapboxgl.Map({ container:'inset', style:'mapbox://styles/mapbox/light-v11', interactive:false });

inset.on('load', ()=>{

// District outline

inset.addSource('blrInset',{type:'geojson',data:blrFC});

inset.addLayer({ id:'blrInset-line', type:'line', source:'blrInset', paint:{ 'line-color':'#1255d7','line-width':2,'line-opacity':0.9 } });

// Basins / Valleys (dashed, labeled)

inset.addSource('basin',{type:'geojson',data:basin});

inset.addLayer({ id:'basin-line', type:'line', source:'basin', paint:{ 'line-color':'#0b5','line-width':2,'line-dasharray':[4,2],'line-opacity':0.9 } });

inset.addLayer({ id:'basin-label', type:'symbol', source:'basin',

layout:{ 'text-field':['coalesce',['get','catchment\_name'],['get','CATCHMENT'],['get','basin'],['get','BASIN']], 'text-size':12, 'text-transform':'uppercase' },

paint:{ 'text-color':'#0b5','text-halo-color':'#fff','text-halo-width':1 } });

// Sub-watersheds (thin dashed, labeled)

inset.addSource('subws',{type:'geojson',data:subws});

inset.addLayer({ id:'subws-line', type:'line', source:'subws', paint:{ 'line-color':'#b45309','line-width':1.2,'line-dasharray':[2,2],'line-opacity':0.9 } });

inset.addLayer({ id:'subws-label', type:'symbol', source:'subws',

layout:{ 'text-field':['coalesce',['get','ws\_Name'],['get','SUBWSHED'],['get','name'],['get','NAME']], 'text-size':11 },

paint:{ 'text-color':'#b45309','text-halo-color':'#fff','text-halo-width':1 } });

// Micro-watersheds (very thin dashed)

inset.addSource('microws',{type:'geojson',data:microws});

inset.addLayer({ id:'microws-line', type:'line', source:'microws', paint:{ 'line-color':'#2563eb','line-width':0.8,'line-dasharray':[1,2],'line-opacity':0.85 } });

// Percolation wells (filtered copy)

inset.addSource('wellsInset',{type:'geojson',data:wellsFiltered});

inset.addLayer({ id:'wellsInset-pts', type:'circle', source:'wellsInset', paint:{ 'circle-radius':3,'circle-color':'#666','circle-stroke-color':'#fff','circle-stroke-width':0.6 } });

inset.fitBounds(turf.bbox(blrFC),{padding:12});

});

});

}

// Keep inset wells in sync

function updateInset(){ if (!inset) return; if (inset.getSource('wellsInset')) inset.getSource('wellsInset').setData(wellsFiltered); }

/\* ---------- Map init ---------- \*/

function initMap(){

map=new mapboxgl.Map({ container:'map', style:'mapbox://styles/mapbox/streets-v12', center:[77.59,12.97], zoom:9.5 });

map.addControl(new mapboxgl.NavigationControl());

map.on('load', async ()=>{

try{

// District (Bengaluru Urban filter)

await getGeo('district');

const blrUrbanFC = { type:'FeatureCollection', features: store.geo.district.features.filter(f => /bengaluru\s\*urban/i.test(getName('district',f.properties))) };

map.addSource('blr',{type:'geojson',data:blrUrbanFC});

map.addLayer({id:'blr-line',type:'line',source:'blr',paint:{'line-color':'#1255d7','line-width':3,'line-opacity':0.9}});

// Zones (hidden until donor/FY or zoom)

await getGeo('zones');

map.addLayer({

id:'zones-fill', type:'fill', source:'zones',

paint:{ 'fill-color':'#cffafe','fill-opacity':0.0 }

});

map.addLayer({

id:'zones-line', type:'line', source:'zones',

paint:{

'line-color':'#0e7490',

'line-width':1.6,

'line-opacity':['step',['zoom'], 0, ZONE\_AUTO\_ZOOM, 0.6, 12, 1.0]

}

});

map.addLayer({

id:'zones-hover', type:'line', source:'zones',

paint:{ 'line-color':'#0b7285','line-width':2.8,'line-opacity':0.9 },

filter:['==',['get','\_\_uid\_\_'],'\_\_none\_\_']

});

// NEW: transparent, fat line so features are always pickable

map.addLayer({

id:'zones-hit', type:'line', source:'zones',

paint:{

'line-color':'rgba(0,0,0,0)', // invisible

'line-width':15, // generous hit area

'line-opacity':0.01 // must be > 0 to render

}

});

// ── Hover + click interactions for zones on the main map ──────────────────────

let hoveredZoneUID = null;

map.on('mousemove', 'zones-hit', (e) => {

const f = e.features && e.features[0];

if (!f) return;

const uid = f.properties && f.properties.\_\_uid\_\_;

if (!uid) return;

// Don’t override an explicit selection (zone dropdown) with hover

const sel = document.getElementById('zoneSel')?.value || '';

if (sel && sel !== '') return;

if (hoveredZoneUID !== uid) {

hoveredZoneUID = uid;

map.setFilter('zones-hover', ['==', ['get','\_\_uid\_\_'], hoveredZoneUID]);

map.getCanvas().style.cursor = 'pointer';

}

});

map.on('mouseleave', 'zones-hit', () => {

hoveredZoneUID = null;

// Clear hover highlight only if no selection is active

const sel = document.getElementById('zoneSel')?.value || '';

if (!sel) map.setFilter('zones-hover', ['==', ['get','\_\_uid\_\_'], '\_\_none\_\_']);

map.getCanvas().style.cursor = '';

});

// Click to SELECT zone (syncs to dropdown and triggers filtering/zoom via existing flows)

map.on('click', 'zones-hit', (e) => {

const f = e.features && e.features[0];

if (!f) return;

const uid = f.properties && f.properties.\_\_uid\_\_;

if (!uid) return;

const zoneSel = document.getElementById('zoneSel');

if (zoneSel) {

// If this zone is already in the dropdown list, set it; otherwise keep current

const inList = [...zoneSel.options].some(o => o.value === uid);

if (inList) {

zoneSel.value = uid;

// This calls rebuildAndFilter('zone') through the existing change handler

const evt = new Event('change', { bubbles: true });

zoneSel.dispatchEvent(evt);

} else {

// No donor/FY restricts list to include this zone; just visually emphasize it

map.setFilter('zones-hover', ['==', ['get','\_\_uid\_\_'], uid]);

}

}

});

// BBMP boundary as union of zones for a clean outline

let bbmpUnion = null;

store.geo.zones.features.forEach(f=>{ try{ bbmpUnion = bbmpUnion ? turf.union(bbmpUnion,f) : f; }catch(\_){} });

const bbmpFC = {type:'FeatureCollection',features: bbmpUnion?[bbmpUnion]:[]};

map.addSource('bbmp',{type:'geojson',data:bbmpFC});

map.addLayer({id:'bbmp-line',type:'line',source:'bbmp',paint:{'line-color':'#e11d48','line-width':3,'line-opacity':0.9}});

// Wards (source only for spatial join)

await getGeo('wards');

// Zone lookups

(function buildZoneLookups(){

const zones = store.geo.zones.features || [];

store.bbmp.zoneNameByUID = {}; store.bbmp.zoneUIDByNormName = {};

zones.forEach(z=>{ const nm = getName('zones', z.properties); store.bbmp.zoneNameByUID[z.properties.\_\_uid\_\_] = nm; store.bbmp.zoneUIDByNormName[norm(nm)] = z.properties.\_\_uid\_\_; if (!z.properties.ZoneName) z.properties.ZoneName = nm; });

})();

// Wells layer (colored by donor)

map.addSource('wells',{type:'geojson',data:wellsFiltered});

map.addLayer({ id:'wells-circle', type:'circle', source:'wells',

paint:{ 'circle-radius': ['interpolate',['linear'],['zoom'], 9,4, 12,7, 14,10], 'circle-stroke-color':'#fff', 'circle-stroke-width':1, 'circle-color':'#999'

}

});

// ── POINT 4 (guarded): Per-well points (hidden until a location is clicked)

if (!map.getSource('wellPoints')) {

map.addSource('wellPoints', { type:'geojson', data: wellPointsAll });

} else {

// If hot-reloaded, just refresh data

map.getSource('wellPoints').setData(wellPointsAll);

}

if (!map.getLayer('wellPoints-circle')) {

map.addLayer({

id:'wellPoints-circle',

type:'circle',

source:'wellPoints',

layout:{ 'visibility':'none' },

paint:{

'circle-radius': 2.5,

'circle-color': '#555',

'circle-opacity': 0.9,

'circle-stroke-color':'#fff',

'circle-stroke-width':0.6

}

});

}

// Bind the well click handler only once

if (!window.\_\_wellPointsClickBound) {

map.on('click','wellPoints-circle',(e)=>{

const p = e.features?.[0]?.properties || {};

const html = `

<div style="font-size:13px;line-height:1.35">

<strong>Well ID:</strong> ${p.well\_id || '—'}<br/>

<strong>Location:</strong> ${p.location || '—'}<br/>

<strong>Donor:</strong> ${p.donor || '—'}<br/>

${p.\_zoneName ? `<strong>Zone/Ward:</strong> ${p.\_zoneName} / ${p.\_wardName || '—'}<br/>` : ''}

</div>`;

new mapboxgl.Popup({maxWidth:'320px'}).setLngLat(e.lngLat).setHTML(html).addTo(map);

});

window.\_\_wellPointsClickBound = true;

}

// Popup: donor summary with compact FY range

// Popup + trigger per-well highlight/zoom for this location

map.on('click','wells-circle', e => {

const f = e.features?.[0]; if (!f) return;

const p = f.properties || {};

const d = p.donor || '';

const stats = d ? donorStats.get(d) : null;

const fyRange = stats ? compactFYRange(stats.years) : '';

const baseHTML = `

<div style="font-size:14px;line-height:1.4">

<strong>Location:</strong> ${p.location||'—'}<br/>

<strong>Donor:</strong> ${p.donor||'—'}<br/>

<strong>Wells:</strong> ${p.wells\_count??'—'}<br/>

<strong>FY:</strong> ${p.fy||'—'}<br/>

<strong>Zone/Ward:</strong> ${p.\_zoneName||'—'} / ${p.\_wardName||'—'}

${(d && stats) ? `

<div style="margin-top:8px;border-top:1px solid #eee;padding-top:6px;">

<span style="font-weight:700">${d}</span> supported

<span class="num" style="font-weight:700">${stats.totalWells}</span> percolation

${plural(stats.totalWells,'well','wells')} across

<span class="num" style="font-weight:700">${stats.totalLocations}</span>

${plural(stats.totalLocations,'location','locations')} from

<span class="num" style="font-weight:700">${fyRange}</span>.

</div>` : ''}

</div>`;

// 🔑 Show/zoom wells for this location; show notice if none

const hadWells = showLocationWells(p.location, f.geometry);

const html = hadWells ? baseHTML : (baseHTML +

`<div style="margin-top:8px;padding-top:6px;border-top:1px solid #eee;color:#6b7280">

Geotagging under progress

</div>`);

new mapboxgl.Popup({maxWidth:'340px'}).setLngLat(e.lngLat).setHTML(html).addTo(map);

});

// Fit to Bengaluru Urban at start

if (blrUrbanFC.features.length){ map.fitBounds(turf.bbox(blrUrbanFC), { padding: 50, duration: 600, maxZoom: 10.8 }); }

// Inset: HYDROLOGY ONLY

buildInset(blrUrbanFC);

// Legend + Filters + Zones visibility reactions

renderLegend();

setupFiltersUI();

map.on('zoomend', updateZoneVisibility);

updateZoneVisibility();

}catch(e){ showError(e.message); }

});

}

/\* ---------- CSV loader (robust to name variants) ---------- \*/

async function fetchPercolationCSV(){

const candidates = [

'Percolation\_ well\_database.csv', // exact filename variant

'Percolation\_%20well\_database.csv',

'Percolation%20well%20database.csv',

'Percolation well database.csv',

'Percolation\_well\_database.csv'

];

for (const path of candidates){ try{ const res = await fetch(path); if(res.ok) return await res.text(); }catch(\_){} }

throw new Error('Could not load the percolation CSV (tried several name variants).');

}

/\* ---------- Load per-well CSV -> GeoJSON + indexes (robust header matching) ---------- \*/

async function fetchWellsCSV(){

const candidates = [ FILES.wells\_pts, 'wells.csv', 'percolation\_wells.csv' ];

for (const path of candidates){ try{ const r = await fetch(path); if (r.ok) return await r.text(); }catch(\_){/\* ignore \*/} }

throw new Error('Could not load the wells CSV (tried several name variants).');

}

function headerPick(cols, matchers){

for (const raw of cols){

const c = stripBOM(raw);

for (const m of matchers){

if (typeof m === 'string'){

if (normHeader(c) === normHeader(m)) return raw; // return original key

} else if (m.test && m.test(c)) {

return raw;

}

}

}

return '';

}

const stripBOM = s => String(s||'').replace(/^\uFEFF/, '');

const normHeader = s => stripBOM(s).toLowerCase().replace(/[\s\_]+/g,' ').trim();

async function loadWellPointsCSV(){

const csvText = await fetchWellsCSV();

const rows = Papa.parse(csvText, { header:true, dynamicTyping:true }).data || [];

if (!rows.length){ wellPointsAll = {type:'FeatureCollection',features:[]}; return; }

const cols = Object.keys(rows[0] || {});

// your columns (robust match)

const donorKey = headerPick(cols, ['Donor Name', /\bdonor\b/i]);

const locNameKey = headerPick(cols, ['Location Name', /\blocation\s\*name\b/i]);

const latWellKey = headerPick(cols, ['Percolation well latitude', /\bpercolation.\*latitude\b/i]);

const lonWellKey = headerPick(cols, ['Percolation well longitude', /\bpercolation.\*long/i]);

// 🔑 robust pick for Well ID (handles BOM / spacing / case)

const wellIdKey = headerPick(cols, [

'Unique ID of the Percolation well',

/unique[\s\_]\*id.\*percolation.\*well/i,

/well.\*unique.\*id/i

]);

// (Optional) quick debug if we didn’t match

if (!wellIdKey) console.warn('Well ID column not found. CSV headers:', cols);

const feats = [];

wellsByLoc.clear();

bboxByLoc.clear();

rows.forEach(r => {

const lat = Number(r[latWellKey]);

const lon = Number(r[lonWellKey]);

if (!Number.isFinite(lat) || !Number.isFinite(lon)) return;

const location = String(r[locNameKey] ?? '').trim();

const donor = String(r[donorKey] ?? '').trim();

// ensure even numeric IDs become strings

const wellId = (r[wellIdKey] ?? '').toString().trim();

const locNorm = norm(location);

const f = {

type:'Feature',

geometry:{ type:'Point', coordinates:[lon, lat] },

properties:{

well\_id: wellId, // <- what the popup will read

location: location,

donor: donor,

\_locNorm: locNorm

}

};

// (optional) zone/ward context (unchanged)

try{

const wards = store.geo.wards?.features || [];

const zones = store.geo.zones?.features || [];

const hitW = wards.find(w => turf.booleanPointInPolygon(f, w));

if (hitW){

f.properties.\_wardName = getName('wards', hitW.properties);

const wardZoneName = hitW.properties.ZoneName || hitW.properties.ZONE\_NAME || hitW.properties.zone || hitW.properties.ZONE || '';

f.properties.\_zoneName = String(wardZoneName).trim();

} else {

const hitZ = zones.find(z => turf.booleanPointInPolygon(f, z));

if (hitZ) f.properties.\_zoneName = getName('zones', hitZ.properties);

}

}catch(\_){}

feats.push(f);

if (!wellsByLoc.has(locNorm)) wellsByLoc.set(locNorm, []);

wellsByLoc.get(locNorm).push(f);

});

wellsByLoc.forEach((arr, key) => {

bboxByLoc.set(key, turf.bbox({ type:'FeatureCollection', features: arr }));

});

wellPointsAll = { type:'FeatureCollection', features: feats };

}

// async function loadWellPointsCSV(){

// const csvText = await fetchWellsCSV();

// const rows = Papa.parse(csvText,{header:true,dynamicTyping:true}).data;

// if (!rows || !rows.length) { wellPointsAll = {type:'FeatureCollection',features:[]}; return; }

// // Heuristics for column names

// const cols = Object.keys(rows[0] || {});

// const latKey = cols.find(c=>/^(lat|latitude)$/i.test(c)) || 'lat';

// const lonKey = cols.find(c=>/^(lon|lng|long|longitude)$/i.test(c)) || 'lon';

// const locKey = cols.find(c=>/location/i.test(c)) || 'Location';

// const wellKey = cols.find(c=>/^(well[\s\_]\*id|unique[\s\_]\*id|id)$/i.test(c)) || 'Well Id';

// const donorKey = cols.find(c=>/^donor/i.test(c)) || 'Donor';

// const fyKey = cols.find(c=>/(fy|financial[\s\_]\*year|year[\s\_]\*of[\s\_]\*completion)/i) || '';

// // Build features

// const feats = [];

// wellsByLoc.clear(); bboxByLoc.clear();

// rows.forEach((r, i) => {

// const lat = Number(r[latKey]), lon = Number(r[lonKey]);

// if (!isFinite(lat) || !isFinite(lon)) return;

// const location = String(r[locKey] ?? '').trim();

// const locNorm = norm(location);

// const wellID = String(r[wellKey] ?? '').trim();

// const donor = String(r[donorKey] ?? '').trim();

// const fy = fyKey ? canonicalFY(r[fyKey]) : '';

// const f = {

// type:'Feature',

// geometry:{ type:'Point', coordinates:[lon, lat] },

// properties:{

// well\_id: wellID,

// location: location,

// donor: donor,

// fy: fy,

// \_locNorm: locNorm

// }

// };

// // Optional: attach zone/ward for context

// const wards = store.geo.wards?.features || [];

// const zones = store.geo.zones?.features || [];

// try{

// const hitW = wards.find(w => turf.booleanPointInPolygon(f, w));

// if (hitW){

// f.properties.\_wardName = getName('wards', hitW.properties);

// const wardZoneName = hitW.properties.ZoneName || hitW.properties.ZONE\_NAME || hitW.properties.zone || hitW.properties.ZONE || '';

// f.properties.\_zoneName = String(wardZoneName).trim();

// } else {

// const hitZ = zones.find(z => turf.booleanPointInPolygon(f, z));

// if (hitZ){

// f.properties.\_zoneName = getName('zones', hitZ.properties);

// }

// }

// }catch(\_){}

// feats.push(f);

// if (!wellsByLoc.has(locNorm)) wellsByLoc.set(locNorm, []);

// wellsByLoc.get(locNorm).push(f);

// });

// // Build per-location bboxes

// wellsByLoc.forEach((arr, key) => {

// bboxByLoc.set(key, turf.bbox({type:'FeatureCollection', features: arr}));

// });

// wellPointsAll = { type:'FeatureCollection', features: feats };

// }

/\* ---------- Show wells for a specific location ---------- \*/

function showLocationWells(locationName, geom){

if (!map.getLayer('wellPoints-circle')) return false;

if (!geom || geom.type !== 'Point') return false;

// Try with 0.4km; if none, expand to 0.8km

const radiiKm = [0.4, 0.8];

for (const r of radiiKm){

const circle = turf.buffer(geom, r, { units: 'kilometers' });

// Count wells inside for zooming; also apply a filter so only those draw

const inside = wellPointsAll.features.filter(f => turf.booleanPointInPolygon(f, circle));

if (inside.length){

// Use a style filter so we don’t rebuild the source

map.setFilter('wellPoints-circle', ['within', circle]);

map.setLayoutProperty('wellPoints-circle', 'visibility', 'visible');

const bb = turf.bbox({ type:'FeatureCollection', features: inside });

map.fitBounds(bb, { padding: 60, duration: 700, maxZoom: 16 });

return true;

}

}

// Nothing found even at 0.8km → hide wells and zoom to the location

map.setLayoutProperty('wellPoints-circle','visibility','none');

map.setFilter('wellPoints-circle', ['==', ['get','\_locNorm'], '\_\_none\_\_']); // harmless reset

const [x,y] = geom.coordinates;

const dLon=0.01, dLat=0.01\*Math.cos(y\*Math.PI/180);

map.fitBounds([[x-dLon,y-dLat],[x+dLon,y+dLat]], { padding:60, duration:600, maxZoom:15 });

return false;

}

/\* ---------- Boot ---------- \*/

(async function start(){

try{

await getGeo('zones'); await getGeo('wards'); await getGeo('district');

const csvText = await fetchPercolationCSV();

const rows = Papa.parse(csvText,{header:true,dynamicTyping:true}).data;

await buildWells(rows);

await loadWellPointsCSV();

console.log('Wells loaded:', wellPointsAll.features.length);

initMap();

}catch(e){ showError(e.message||'Startup error'); }

})();

</script>

</body>

</html>