計画行政学会 GIS 分科会 可視化セミナー

WebGISはじめの一歩、自分でコントロールする可視化基盤

資料：https://alt9800.github.io/Seminars/

アジェンダ

- Git / Githubの説明
- Webプログラミング入門
- ソースコードの公開とは？

上記の内容(前回)を踏まえて、今回は、

WebGISはじめの一歩、自分でコントロールする可視化基盤

についてお話しします。

~準備~

必要なもの

テキストエディタ（メモ帳でもOK、VS Codeがあればベター）
ブラウザ（Chrome/Firefox/Edge など）

ホスティングをするためにサーバーが必要です。
プラグインLive-serverや`http-server`を利用できるようになると簡単です。
(インターネットに成果を公開したい場合はGitHub Pagesでホスティングするとよい)

ファイル作成

デスクトップなどに`handson`フォルダを作成し、その中に`leaflet`
そのなかに、`index.html`を作成してください。

Leaflet編

Step1 地図を表示させる

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Leaflet 基本の地図</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- Leaflet CSS -->
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />

    <style>
        body { margin: 0; padding: 0; }
        #map { width: 100%; height: 100vh; }
    </style>
</head>
<body>
    <div id="map"></div>

    <!-- Leaflet JavaScript -->
    <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>

    <script>
        // 地図の初期化（東京駅中心）
        const map = L.map('map').setView([35.6812, 139.7671], 13);

        // 背景地図タイルの追加
        L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
            attribution: '&copy; OpenStreetMap contributors'
        }).addTo(map);
    </script>
</body>
</html>
```

以下にアクセス

http://localhost:8080/leaflet/

(使用するホスティングサーバによって、ポート番号が`5000`などであることも)

Step 2: マーカーを追加

</script> の直前に以下を追加：

```
// 東京駅にマーカーを追加
        const marker = L.marker([35.6812, 139.7671]).addTo(map);
        marker.bindPopup('<b>東京駅</b><br>ここが東京の中心です').openPopup();
```

Step 3: 複数のマーカーを追加

以下を削除。

```
// 東京駅にマーカーを追加
        const marker = L.marker([35.6812, 139.7671]).addTo(map);
        marker.bindPopup('<b>東京駅</b><br>ここが東京の中心です').openPopup();
```

以下に置き換え。

```
// 東京の主要駅データ
        const stations = [
            { name: '東京駅', lat: 35.6812, lng: 139.7671 },
            { name: '新宿駅', lat: 35.6896, lng: 139.7006 },
            { name: '渋谷駅', lat: 35.6580, lng: 139.7016 },
            { name: '品川駅', lat: 35.6284, lng: 139.7387 },
            { name: '池袋駅', lat: 35.7295, lng: 139.7109 }
        ];

        // 各駅にマーカーを配置
        stations.forEach(function(station) {
            L.marker([station.lat, station.lng])
                .addTo(map)
                .bindPopup('<b>' + station.name + '</b>');
        });
```

Step 4: 円を描く

```
// 東京駅から半径5kmの円
L.circle([35.6812, 139.7671], {
    color: 'red',
    fillColor: '#f03',
    fillOpacity: 0.2,
    radius: 5000
}).addTo(map).bindPopup('東京駅から半径5km');
```

Step 5: 背景地図を変更

国土地理院の地図に変更：

```
L.tileLayer('https://cyberjapandata.gsi.go.jp/xyz/std/{z}/{x}/{y}.png', {
            attribution: '&copy; <a href="https://maps.gsi.go.jp/development/ichiran.html">国土地理院</a>'
        }).addTo(map);
```

航空写真に変更：

```
L.tileLayer('https://cyberjapandata.gsi.go.jp/xyz/seamlessphoto/{z}/{x}/{y}.jpg', {
            attribution: '&copy; <a href="https://maps.gsi.go.jp/development/ichiran.html">国土地理院</a>'
        }).addTo(map);
```

チャレンジ

- 自分の住んでいる地域の座標に変更してみよう
- マーカーの色を変えてみよう
- 線を引いてみよう（L.polyline()）

Leafletを使う場合はこちらでもいい(QGISの履修が前提のため)

qgis2web — QGIS Python Plugins Repository
https://plugins.qgis.org/plugins/qgis2web/

- バッファーやボロノイを作る場合などは座標系の変換に注意
  (ユークリッド距離との変換が大変)
- OpenLeyersなども利用できる

Leafletは書き方が変わります！！

https://leafletjs.com/2025/05/18/leaflet-2.0.0-alpha.html

細かいお話だと、Python(JupyterNotebookやColaboratory含む)から`foloum`や`leafmap(ipyLeaflet)`を利用して地図レイヤーを読み込む際には、
ブラウザを介してLeafletを使ってデータの可視化を行なっています。

MapLibre編

Step 1: 基本の地図を表示

handson/maplibre/index.html を作成し、以下をコピペ：

MapLibreではデフォルトでベクター形式の背景地図が用意されています。(Demo Tiles)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>MapLibre 基本の地図</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- MapLibre CSS -->
    <link rel="stylesheet" href="https://unpkg.com/maplibre-gl@4.7.1/dist/maplibre-gl.css" />

    <style>
        body { margin: 0; padding: 0; }
        #map { width: 100%; height: 100vh; }
    </style>
</head>
<body>
    <div id="map"></div>

    <!-- MapLibre JavaScript -->
    <script src="https://unpkg.com/maplibre-gl@4.7.1/dist/maplibre-gl.js"></script>

    <script>
        // 地図の初期化（東京駅中心）
        const map = new maplibregl.Map({
            container: 'map',
            style: 'https://demotiles.maplibre.org/style.json',
            center: [139.7671, 35.6812],
            zoom: 13
        });
```

Step 2: 国土地理院の地図に変更

MapLibreでは「スタイルJSON」で地図のデザインを定義します。
styleの部分を以下に置き換え：

```
const map = new maplibregl.Map({
            container: 'map',
            style: {
                version: 8,
                sources: {
                    'gsi-std': {
                        type: 'raster',
                        tiles: [
                            'https://cyberjapandata.gsi.go.jp/xyz/std/{z}/{x}/{y}.png'
                        ],
                        tileSize: 256,
                        attribution: '&copy; <a href="https://maps.gsi.go.jp/development/ichiran.html">国土地理院</a>'
                    }
                },
                layers: [
                    {
                        id: 'gsi-std-layer',
                        type: 'raster',
                        source: 'gsi-std',
                        minzoom: 0,
                        maxzoom: 18
                    }
                ]
            },
            center: [139.7671, 35.6812],
```

💡ポイント

MapLibreの最大の強みは、ベクター形式でタイルを読み込める点です。
ラスタータイルはあらかじめpngなどで矩形(メッシュ・グリッド)に分けた領域を生成しておく一方で、
ベクタータイルはクライアントサイドで地図のスタイルをある程度制御でき便利です。

■各タイルがどのように配信されているか
https://wiki.openstreetmap.org/wiki/Japan/OSMFJ_Tileserver
■ベクタータイルのスタイルを実際に見てみましょう
https://tile.openstreetmap.jp/styles/osm-bright-ja/style.json

様々な背景レイヤを切り替える例は
(ext01)maplibreSwitchの項目を参照してください。

Step 3: マーカーを追加

ナビゲーションコントロールの後に追加：

```
// 地図が読み込まれたら実行
        map.on('load', function() {
            // 東京の主要駅データ
            const stations = [
                { name: '東京駅', lng: 139.7671, lat: 35.6812 },
                { name: '新宿駅', lng: 139.7006, lat: 35.6896 },
                { name: '渋谷駅', lng: 139.7016, lat: 35.6580 },
                { name: '品川駅', lng: 139.7387, lat: 35.6284 },
                { name: '池袋駅', lng: 139.7109, lat: 35.7295 }
            ];

            // 各駅にマーカーを配置
            stations.forEach(function(station) {
                // ポップアップを作成
                const popup = new maplibregl.Popup({ offset: 25 })
                    .setHTML('<b>' + station.name + '</b>');

                // マーカーを作成
                new maplibregl.Marker()
                    .setLngLat([station.lng, station.lat])
                    .setPopup(popup)
                    .addTo(map);
            });
```

Step 4: 円を描く

以下のようになっている場所の後`map.on('load', function () {`に中心点を生成し、そこからバッファを作成します。

```
        // ナビゲーションコントロール（ズームボタン）を追加
        map.addControl(new maplibregl.NavigationControl());
        // 地図が読み込まれたら実行
        map.on('load', function () {
```

`map.on('load', function () { });`の中に収まっているか要確認！

次のページ👇

```
// 円のポイントを生成する関数
            function createCircle(center, radiusInKm, points = 64) {
                const coords = {
                    latitude: center[1],
                    longitude: center[0]
                };

                const km = radiusInKm;
                const ret = [];
                const distanceX = km / (111.320 * Math.cos(coords.latitude * Math.PI / 180));
                const distanceY = km / 110.574;

                for (let i = 0; i < points; i++) {
                    const theta = (i / points) * (2 * Math.PI);
                    const x = distanceX * Math.cos(theta);
                    const y = distanceY * Math.sin(theta);
                    ret.push([coords.longitude + x, coords.latitude + y]);
                }
                ret.push(ret[0]); // 円を閉じる

                return ret;
            }

            // 円データソースを追加
            map.addSource('tokyo-circle', {
                type: 'geojson',
                data: {
                    type: 'Feature',
                    geometry: {
                        type: 'Polygon',
                        coordinates: [createCircle([139.7671, 35.6812], 5)]
                    }
                }
            });

            // 円の塗りつぶし
            map.addLayer({
                id: 'circle-fill',
                type: 'fill',
                source: 'tokyo-circle',
                paint: {
                    'fill-color': '#ff3333',
                    'fill-opacity': 0.3
                }
            });

            // 円の枠線
            map.addLayer({
                id: 'circle-outline',
                type: 'line',
                source: 'tokyo-circle',
                paint: {
                    'line-color': '#ff0000',
                    'line-width': 2
                }
            });
```

☝️ワンポイント

少し今風のJavascriptの書き方をしています。

```
function onMapLoad() {
    // ...
}

map.on('load', onMapLoad);
```

のように、関数を設定しておいて、地図の読み込みが完了したら(`load`)、`map`オブジェクトにおいて、`.on`メソッドとして`onMapLoad`関数を呼び出す(実行する)、という流れなのですが、

```
map.on('load', function() {
    // ...
});
```

このような書き方をすることで、無名の関数として、実行したい処理を直後に書くこ

ここまででピンとくる方もいらっしゃると思いますが、
データソース(多くはgeojson形式)は、htmlやjsの中に記載することもできますし、
外部ファイルとして読み込むこともできます。

Step 5: 3D建物を表示

`circle` レイヤーの後に追加：
前ステップと同様に `map.on('load', function () { });` の中に収まっていることを確認！

```
// 簡易的な建物データを作成（東京駅周辺の架空のビル）
            map.addSource('buildings', {
                type: 'geojson',
                data: {
                    type: 'FeatureCollection',
                    features: [
                        {
                            type: 'Feature',
                            properties: {
                                name: '高層ビルA',
                                height: 150,
                                base_height: 0
                            },
                            geometry: {
                                type: 'Polygon',
                                coordinates: [[
                                    [139.765, 35.680],
                                    [139.767, 35.680],
                                    [139.767, 35.682],
                                    [139.765, 35.682],
                                    [139.765, 35.680]
                                ]]
                            }
                        },
                        {
                            type: 'Feature',
                            properties: {
                                name: '高層ビルB',
                                height: 200,
                                base_height: 0
                            },
                            geometry: {
                                type: 'Polygon',
                                coordinates: [[
                                    [139.768, 35.681],
                                    [139.770, 35.681],
                                    [139.770, 35.683],
                                    [139.768, 35.683],
                                    [139.768, 35.681]
                                ]]
                            }
                        },
                        {
                            type: 'Feature',
                            properties: {
                                name: '中層ビルC',
                                height: 80,
                                base_height: 0
                            },
                            geometry: {
                                type: 'Polygon',
                                coordinates: [[
                                    [139.764, 35.683],
                                    [139.766, 35.683],
                                    [139.766, 35.684],
                                    [139.764, 35.684],
                                    [139.764, 35.683]
                                ]]
```

```
            // 円の塗りつぶし
            map.addLayer({
            });
            // 円の枠線
            map.addLayer({
            });
```

これらを折りたたむとわかりやすいかも。

これは、MapLibre GL JSのコンストラクタの引数に、`pitch`と`bearing`を追加すると見やすいかも。

```
        const map = new maplibregl.Map({
            container: 'map',
            style: {
                // ... 省略 ...
            },
            center: [139.7671, 35.6812],
            zoom: 14,      // ズームを上げる
            pitch: 60,     // 傾きを追加
            bearing: 0
        });
```

ここまでのコードをmaplibre-compにまとめてあります。
確認してみましょう。

💡オープンデータ色々

https://nlftp.mlit.go.jp/ksj/gml/datalist/KsjTmplt-N03-2024.html

```
提案1：東京23区の境界データ（ポリゴン）
javascript// 国土数値情報から取得可能
const tokyoWards = 'https://raw.githubusercontent.com/dataofjapan/land/master/tokyo23.geojson';
→ コロプレスマップ（色分け地図）を作成
提案2：東京の鉄道路線（ライン）
javascript// 国土数値情報の鉄道データ
const railwayLines = 'https://nlftp.mlit.go.jp/ksj/gml/data/N02/N02-23/N02-23_GML.zip';
→ 路線図の可視化
提案3：避難所データ（ポイント）
javascript// 国土数値情報の避難所データ
const shelters = 'https://nlftp.mlit.go.jp/ksj/gml/data/P20/P20-12/P20-12_13_GML.zip';
→ ポイントマップ、ヒートマップ
```

データ読み込みの例

(ext02)maplbreLine の項目を参照

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>MapLibre 路線表示</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://unpkg.com/maplibre-gl@4.7.1/dist/maplibre-gl.css" />
    <style>
        body { margin: 0; padding: 0; }
        #map { width: 100%; height: 100vh; }
    </style>
</head>
<body>
    <div id="map"></div>
    <script src="https://unpkg.com/maplibre-gl@4.7.1/dist/maplibre-gl.js"></script>
    <script>
        const map = new maplibregl.Map({
            container: 'map',
            style: {
                version: 8,
                sources: {
                    'gsi-pale': {
                        type: 'raster',
                        tiles: ['https://cyberjapandata.gsi.go.jp/xyz/pale/{z}/{x}/{y}.png'],
                        tileSize: 256
                    }
                },
                layers: [{
                    id: 'gsi-pale-layer',
                    type: 'raster',
                    source: 'gsi-pale'
                }]
            },
            center: [139.62, 35.67],
            zoom: 11
        });

        map.on('load', function() {
            fetch('../data/railway_simple.geojson')
                .then(response => response.json())
                .then(data => {
                    map.addSource('railway', {
                        type: 'geojson',
                        data: data
                    });

                    map.addLayer({
                        id: 'railway-line',
                        type: 'line',
                        source: 'railway',
                        paint: {
                            'line-color': '#FF6347',
                            'line-width': 4
                        }
                    });
                });
```

さらにおまけ

ext03 : ヒートマップ機能

ext04 : クラスター機能

を示しています。

Asahinaさんのハンズオン資料がとてもよくできていておすすめ！

https://zenn.dev/asahina820/books/c29592e397a35b

(OSGeoの私の上役ですが...)

DeckGL編

Step 1: 基本の地図を表示（OSMタイル）

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Deck.gl 基本の地図</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        body { margin: 0; padding: 0; font-family: Arial, sans-serif; }
        #map { width: 100%; height: 100vh; position: relative; }
    </style>
</head>
<body>
    <div id="map"></div>

    <script src="https://unpkg.com/deck.gl@9.0.0/dist.min.js"></script>

    <script>
        // Deck.glの初期化
        const deckgl = new deck.DeckGL({
            container: 'map',
            initialViewState: {
                longitude: 139.65,
                latitude: 35.67,
                zoom: 11,
                pitch: 0,
                bearing: 0
            },
            controller: true,
            layers: [
                // 背景地図タイル（OpenStreetMap）
                new deck.TileLayer({
                    id: 'osm-tiles',
                    data: 'https://tile.openstreetmap.org/{z}/{x}/{y}.png',
                    minZoom: 0,
                    maxZoom: 19,
                    tileSize: 256,
                    renderSubLayers: props => {
                        const {
                            bbox: {west, south, east, north}
                        } = props.tile;

                        return new deck.BitmapLayer(props, {
                            data: null,
                            image: props.data,
                            bounds: [west, south, east, north]
                        });
                    }
                })
            ]
        });
    </script>
</body>
</html>
```

Step 2: 背景地図切り替え機能を追加

`<div id="map"></div>` の下に以下を追加：

```
<div id="controls">
        <strong>背景地図切り替え</strong>
        <button onclick="changeBaseMap('osm')">OpenStreetMap</button>
        <button onclick="changeBaseMap('gsi')">国土地理院</button>
        <button onclick="changeBaseMap('carto')">Carto Light</button>
</div>
```

`<style>` タグ内に以下を追加：

```
#controls {
            position: absolute;
            top: 10px;
            left: 10px;
            background: white;
            padding: 10px;
            border-radius: 5px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.3);
            z-index: 1;
        }
        #controls button {
            display: block;
            margin: 5px 0;
            padding: 8px 12px;
            cursor: pointer;
            border: none;
            background: #007cbf;
            color: white;
            border-radius: 3px;
        }
        #controls button:hover {
            background: #005a8c;
        }
```

`const deckgl = new deck.DeckGL({` の前に以下を追加：

```
// 背景地図のタイル設定
        const baseMaps = {
            osm: 'https://tile.openstreetmap.org/{z}/{x}/{y}.png',
            gsi: 'https://cyberjapandata.gsi.go.jp/xyz/pale/{z}/{x}/{y}.png',
            carto: 'https://basemaps.cartocdn.com/light_all/{z}/{x}/{y}.png'
        };

        let deckgl;

        // タイルレイヤーを作成する関数
        function createTileLayer(mapKey) {
            return new deck.TileLayer({
                id: 'tile-layer',
                data: baseMaps[mapKey],
                minZoom: 0,
                maxZoom: 19,
                tileSize: 256,
                renderSubLayers: props => {
                    const {
                        bbox: {west, south, east, north}
                    } = props.tile;

                    return new deck.BitmapLayer(props, {
                        data: null,
                        image: props.data,
                        bounds: [west, south, east, north]
                    });
                }
            });
        }

        // 背景地図を切り替える関数
        function changeBaseMap(mapKey) {
            deckgl.setProps({
                layers: [createTileLayer(mapKey)]
            });
        }
```

そして `const deckgl = new deck.DeckGL({` を以下に変更：

```
deckgl = new deck.DeckGL({
```

⚠️ライセンス表記を忘れずに！

```
<style>
    .map-attribution {
        position: absolute;
        bottom: 0;
        right: 0;
        background: rgba(255, 255, 255, 0.9);
        padding: 3px 8px;
        font-size: 9px;
        font-family: Arial, sans-serif;
        z-index: 1000;
        border-top-left-radius: 3px;
    }
    .map-attribution a {
        color: #0078A8;
        text-decoration: none;
        margin: 0 3px;
    }
</style>

<div class="map-attribution">
        © <a href="https://www.openstreetmap.org/copyright" target="_blank">OpenStreetMap Contributors</a> |
        © <a href="https://maps.gsi.go.jp/development/ichiran.html" target="_blank">GSI</a> |
        © <a href="https://carto.com/attributions" target="_blank">CARTO</a>
</div>
```

Step 3: ScatterplotLayerで点を表示

`changeBaseMap` 関数の前に以下を追加：

```
// 主要駅データ
        const stationsData = [
            { name: '東京駅', coordinates: [139.7671, 35.6812], passengers: 462000 },
            { name: '新宿駅', coordinates: [139.7006, 35.6896], passengers: 775000 },
            { name: '渋谷駅', coordinates: [139.7016, 35.6580], passengers: 379000 },
            { name: '品川駅', coordinates: [139.7387, 35.6284], passengers: 379000 },
            { name: '池袋駅', coordinates: [139.7109, 35.7295], passengers: 558000 }
        ];
```

`changeBaseMap` 関数を以下に変更：

```
function changeBaseMap(mapKey) {
            deckgl.setProps({
                layers: [
                    createTileLayer(mapKey),
                    // ScatterplotLayer - 駅のポイント
                    new deck.ScatterplotLayer({
                        id: 'stations',
                        data: stationsData,
                        getPosition: d => d.coordinates,
                        getRadius: d => Math.sqrt(d.passengers) * 2,
                        getFillColor: [255, 140, 0],
                        pickable: true,
                        radiusMinPixels: 5,
                        radiusMaxPixels: 50
                    })
                ]
            });
        }
```

そして初期表示用に `deckgl = new deck.DeckGL({` の `layers:` 部分を以下に変更：

```
layers: [
                createTileLayer('osm'),
                new deck.ScatterplotLayer({
                    id: 'stations',
                    data: stationsData,
                    getPosition: d => d.coordinates,
                    getRadius: d => Math.sqrt(d.passengers) * 2,
                    getFillColor: [255, 140, 0],
                    pickable: true,
                    radiusMinPixels: 5,
                    radiusMaxPixels: 50
                })
            ]
```

Step 4: ツールチップを追加

`deckgl = new deck.DeckGL({` の設定に以下を追加：

```
getTooltip: ({object}) => object && object.name && {
                html: `<strong>${object.name}</strong><br/>乗降客数: ${object.passengers.toLocaleString()}人/日`,
                style: {
                    backgroundColor: '#333',
                    color: '#fff',
                    padding: '10px',
                    borderRadius: '5px'
                }
            },
```

具体的には

```
        // Deck.glの初期化
        deckgl = new deck.DeckGL({
            container: 'map',
            initialViewState: {
                longitude: 139.65,
                latitude: 35.67,
                zoom: 11,
                pitch: 0,
                bearing: 0
            },
            controller: true,
```

の後に併記します。

```
            new deck.ScatterplotLayer({
                id: 'stations',
                data: stationsData,
                getPosition: d => d.coordinates,
                getRadius: d => Math.sqrt(d.passengers) * 2,
                getFillColor: [255, 140, 0],
                pickable: true,
                radiusMinPixels: 5,
                radiusMaxPixels: 50,
                autoHighlight: true,
                highlightColor: [255, 255, 0, 150]
            })
```

みたいにハイライトを与えるとインタラクティブでGood。

Step 5: GeoJSON形式でパスを表示

`stationsData` の後に以下を追加：

```
// GeoJSON形式のパスデータ
        const pathData = {
            type: 'FeatureCollection',
            features: [
                {
                    type: 'Feature',
                    properties: { name: '中央線', color: [255, 99, 71] },
                    geometry: {
                        type: 'LineString',
                        coordinates: [
                            [139.7003573288469, 35.691239928314374],
                            [139.7009635407017, 35.68820849667566],
                            [139.69578888654496, 35.68369419472977],
                            [139.6875474637206, 35.67832317143727],
                            [139.68017274365303, 35.67583437982546],
                            [139.6676212831522, 35.6734214567513],
                            [139.65915365759162, 35.67072037158694],
                            [139.6510055510484, 35.668539799401145],
                            [139.64190275546554, 35.66589306040666],
                            [139.6326221009918, 35.66738232477873],
                            [139.6240861011417, 35.668152685857876],
                            [139.61514396851464, 35.669913763965226],
                            [139.60986015831952, 35.67079428322272],
                            [139.601331512042, 35.66793266851582],
                            [139.58480410266674, 35.662319842064704],
                            [139.57538175409587, 35.65830669802037],
                            [139.5669694195912, 35.65422992235908],
                            [139.55914740892905, 35.65035573108334],
                            [139.55210343625362, 35.64969588326824],
                            [139.54438400157323, 35.65167794492214]
                        ]
                    }
```

初期表示の `layers` を更新：

```
layers: [
                createTileLayer('osm'),
                // PathLayer - GeoJSON形式の路線
                new deck.PathLayer({
                    id: 'path-layer',
                    data: pathData.features,
                    getPath: d => d.geometry.coordinates,
                    getColor: d => d.properties.color,
                    getWidth: 40,
                    widthMinPixels: 2,
                    pickable: true
                }),
                new deck.ScatterplotLayer({
                    id: 'stations',
                    data: stationsData,
                    getPosition: d => d.coordinates,
                    getRadius: d => Math.sqrt(d.passengers) * 2,
                    getFillColor: [255, 140, 0],
                    pickable: true,
                    radiusMinPixels: 5,
                    radiusMaxPixels: 50,
                    autoHighlight: true,
                    highlightColor: [255, 255, 0, 150]
                })
            ]
```

`changeBaseMap` 関数も更新：

```
function changeBaseMap(mapKey) {
            deckgl.setProps({
                layers: [
                    createTileLayer(mapKey),
                    new deck.PathLayer({
                        id: 'path-layer',
                        data: pathData.features,
                        getPath: d => d.geometry.coordinates,
                        getColor: d => d.properties.color,
                        getWidth: 40,
                        widthMinPixels: 2,
                        pickable: true
                    }),
                    new deck.ScatterplotLayer({
                        id: 'stations',
                        data: stationsData,
                        getPosition: d => d.coordinates,
                        getRadius: d => Math.sqrt(d.passengers) * 2,
                        getFillColor: [255, 140, 0],
                        pickable: true,
                        radiusMinPixels: 5,
                        radiusMaxPixels: 50,
                        autoHighlight: true,
                        highlightColor: [255, 255, 0, 150]
                    })
                ]
            });
        }
```

応用編

グリッド押出: ext-01

ヘキサゴン: ext-02

アーク: ext-03

ヒートマップ: ext-04

他のサンプルをぜひ確認してみて下さい。

https://deck.gl/examples

ここまでの知識を整理すると、

可視化そのものは道具でしかなくて、その前処理としてデータをどのように整理するか、という部分で構造的なプログラミングが必要になることも多いです。
この部分もLLMに頼ると良いと思います。

これらの知識を踏まえて、
MapLibre と DeckGLを高度に統合し、大規模なデータを読み込むことをサポートした KeplerGLを利用するところから始めると良いでしょう。

このシステム自体もOSSなので、ご自身の管理環境にデプロイすることもできますし、背景地図をオープンなものに置き換えるとスクリーンショットを成果として活用しやすいと思います。

そのほかのTips

白地図が欲しい

Natural Earth のデータを加工し、Geojsonとして表示し、背景レイヤーをOFFにする

あるいはSVGとして持ち、D3.jsで表示する方法もある

高度な3D表現

Cesium.jsを使う方法もあります。

ストーリーテリング

Re:Eeathなどが便利です。

次回予告

福井県立大学 地域経済研究所
青木和人先生

公共オープンデータ利活用の勘所

国や自治体からはどのようなデータが公開されていますが、どのようなプラットフォームで公開されているのか、あるいはどのような性質のデータが多いのかをあらかじめ知っておけばデータ選定の効率化が行えます。
現時点でどんなデータが出ているかを踏まえて、利活用事例についてお伝えし、
オープンデータが公開されることによる受益者をどう設定するべきか、政策立案にどのように繋げるかの糸口をこれまでの活動を通してお伝えします。

12月13日土曜日 18:00 -
60 ~ 90 分ほどを予定しています。

場所 計画行政学会GIS研究会 Discord (https://discord.gg/aThG5Vge9g)