Web Map of Nepal with Multiple Layers and Real-Time Data

This web map integrates multiple dynamic layers, including administrative boundaries (districts, provinces, local units), real-time data (flood discharge, wind speed), and points of interest (schools, national parks, and nearby places). **Leaflet is used** for design of interactive mapping, **GeoJSON** for data visualization, and APIs such as **Open-Meteo** and **Foursquare** is included in this map. This map enables users to explore geographical and environmental data in a highly interactive way.

Adding Base Map and setting up the map:

Following **the Leaflet Quick Start Guide**, first of all Leaflet CSS and Javascript file were added. A div element is used for the setting up the map on our webpage.

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"/>

<script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
```

```
<div id="map" style="height: 500px; width: 100%;"></div>
```

A map showing the extent of Nepal is adjusted using the following code. The code's coordinates are adjusted in such a way that Nepal is extended over our screen.

```
const map = L.map ('map', {center: [28.345, 84.134], zoom: 7});
```

Then the base layer is added. Open street map is considered to be the basemap. Basemap provides the foundation to work on the map for the visualization propose.

```
const osm = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
maxZoom: 20,
attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>
}).addTo(map);
```

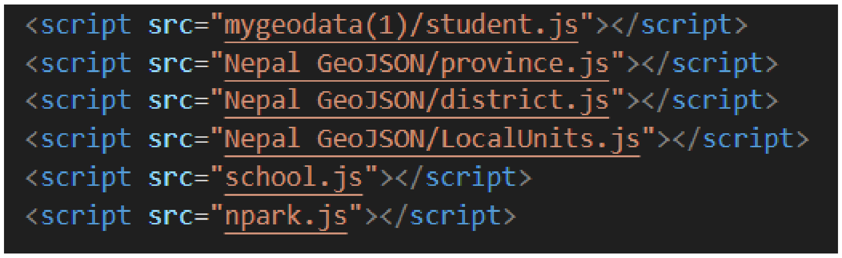
To enhance our basemap we can add zoom controls and scale. We will have zoom in and out button on the button right hand side of the map and scale changes accordingly to the zoom level.

```
L.control.zoom({ position: 'bottomright' }).addTo(map);
L.control.scale().addTo(map);
```

Here is the final map upto now, It show the base layer, scale and zoom controls options. Wile opening the webpage the extent will be on Nepal.

Adding GeoJson

To add the GeoJson file first of all we copied .geojson to make it .js file. The variable or constants of the same js file contains the code of .geojson. The .js file must be added as a base before working with it. We basically cannot add other format of file like .shp directly using leaflet so all sorts of files must be converted to geojson.



```
<script src="mygeodata(1)/student.js"></script>
<script src="Nepal GeoJSON/province.js"></script>
<script src="Nepal GeoJSON/district.js"></script>
<script src="Nepal GeoJSON/LocalUnits.js"></script>
<script src="school.js"></script>
<script src="npark.js"></script>
```

Figure 1: .geojson converted to .js and added as a base.

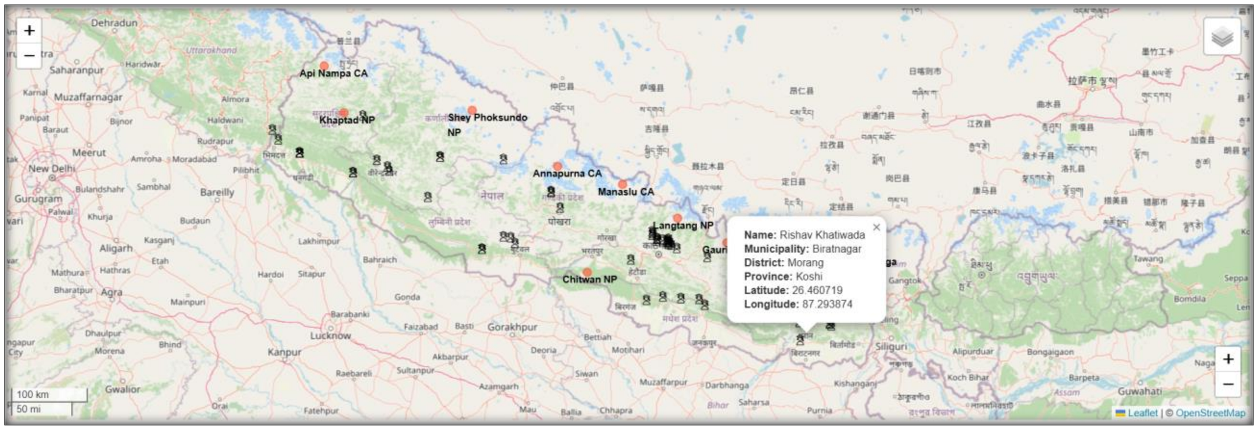
GE20 Student's location

First of all, the data of every student of Geomatics Engineering at Kathmandu University- Batch 2020 was collected. The collected data was in excel file and it was converted to GeoJSON using *https://mapshaper.org/* website. As mentioned above after converting the geojson to js , the js file was added as student.js and the following code from leaflet is used to get the desired output.

In map there exists icon for each student location. On clicking the icon the properties of student like Name, Municipality, province, and district pops up.

```
const studenticon = L.icon({
        iconUrl: 'mystudent.svg',
        iconSize: [16, 16],
        iconAnchor: [16, 32],
        popupAnchor: [0, -32]
                });
        const studenthome = L.geoJSON(studentdata, {
            pointToLayer: function (features, latlng) {
                return L.marker(latlng, { icon: studenticon });},
            onEachFeature: function (feature, layer) {
                 layer.bindPopup(properties....)}
        }).addTo(map);
```

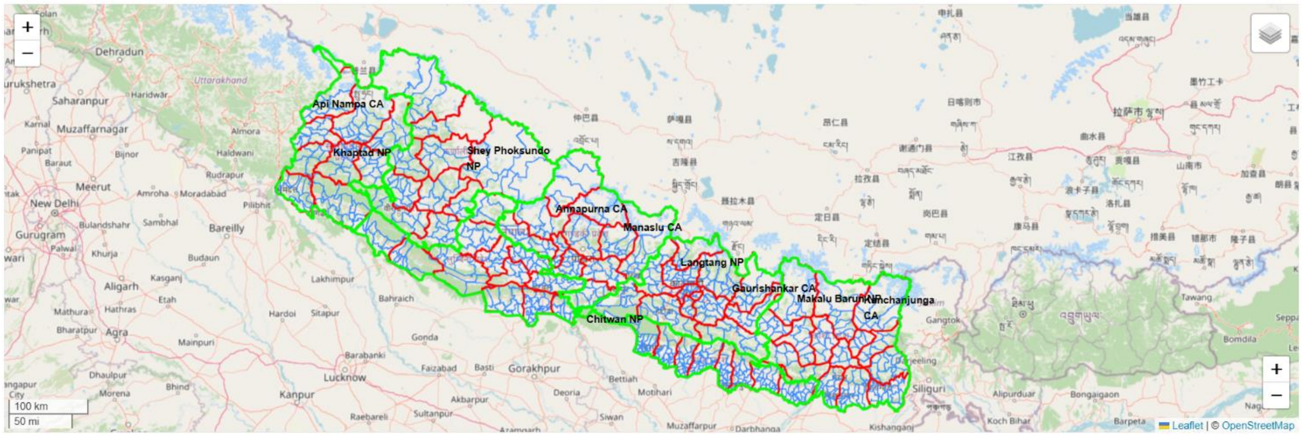
L.geoJson is used as described in Leaflet library. The icon is customized and the properties mentioned on the code are displayed.


Name: Rishav Khatiwada
Municipality: Biratnagar
District: Morang
Province: Koshi
Latitude: 26.460719
Longitude: 87.293874

Adding Province, District and Municipality Administrative Boundary

As previous first of all geojson file of each three were downloaded from online portals. Following the same procedure (converting to .js and adding it as a base), province, district and municipality are shown on the map. The style option was used to displayed them distinctively.

```
          // Adding District GeoJson
        const districtNepal = L.geoJSON(district, { style: { color: '#FF0000', fill: false, weight: 2 } });
        // Adding Province GeoJson
        const provinceNepal = L.geoJSON(province, { style: { color: '#00ff00', fill: false, weight: 3 } });
        // Adding Local Units GeoJson
        const localunits = L.geoJSON(LocalUnits, { style: { fill: false, opacity: 2, weight: 1 } });
```



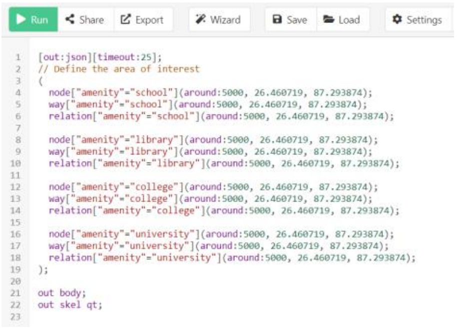
Using OverPass Turbo

The tool is particularly helpful for extracting specific data from OSM such as roads, buildings, or natural features, and viewing them geographically. Some key features of Overpass Turbo include the ability to manipulate queries using special symbols and syntax, visualize geometric data, and share queries via permanent links. It also supports integration with bounding boxes and automatic completion to improve the accuracy and relevance of data requests.

School Around me around 5000 meter:

The data of school around my location (which we have already added for each students) is extracted from OSM. We used overpass turbo to run the query for adding the buffer of 5000m and include all the school around the area.

I have selected educational institutions like school, library, college, and university around 5000m around the student having latitude and longitude 26.460719,87.293874.
The data is exported in geojson format and the same procedures was followed to add this extracted geojson file using leaflet.



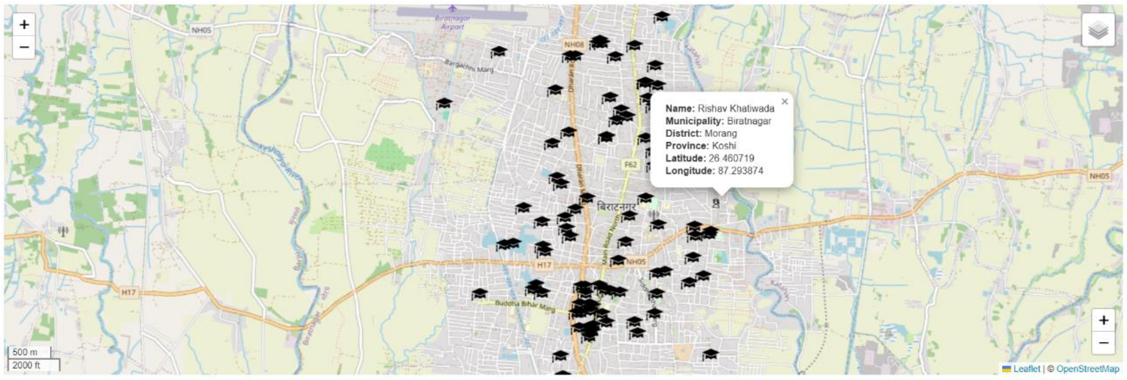
```
[out:json][timeout:25];
// Define the area of interest
(
  node["amenity"="school"](around:5000, 26.460719, 87.293874);
  way["amenity"="school"](around:5000, 26.460719, 87.293874);
  relation["amenity"="school"](around:5000, 26.460719, 87.293874);

  node["amenity"="library"](around:5000, 26.460719, 87.293874);
  way["amenity"="library"](around:5000, 26.460719, 87.293874);
  relation["amenity"="library"](around:5000, 26.460719, 87.293874);

  node["amenity"="college"](around:5000, 26.460719, 87.293874);
  way["amenity"="college"](around:5000, 26.460719, 87.293874);
  relation["amenity"="college"](around:5000, 26.460719, 87.293874);

  node["amenity"="university"](around:5000, 26.460719, 87.293874);
  way["amenity"="university"](around:5000, 26.460719, 87.293874);
  relation["amenity"="university"](around:5000, 26.460719, 87.293874);
);

out body;
out skel qt;
```



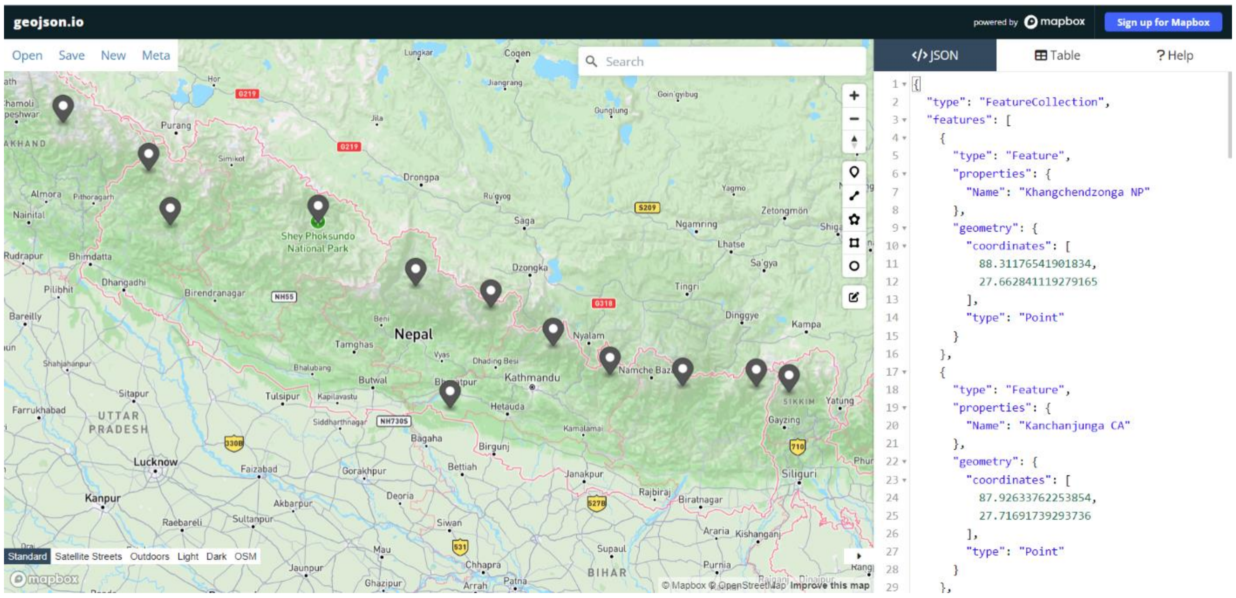
Name: Rishav Khatiwada
Municipality: Biratnagar
District: Morang
Province: Koshi
Latitude: 26.460719
Longitude: 87.293874

Using GeoJson.io

GeoJSON.io is a web-based platform designed for the creation, editing, and sharing of GeoJSON files which are widely used in mapping and spatial data analysis. This tool allows users to visualize geographic data on an interactive map and easily generate or modify GeoJSON data without the need for coding expertise. Users can add points, lines, and polygons directly onto the map, and their actions are automatically converted into GeoJSON format.

Adding National park and Conservation Area geojson using GeoJson.io

The json file was creating by digitizing the map available on the site. The automatic code is generated which built up the json file. Inside property name of point feature was labeled. The json file was exported and added using leaftet. The purpose of adding those detail on the map is to show the label of the data. We can see our map contains the label of conservation area and national park from beginning of the screenshots. Label of those detail is overlayed above all the existing layers and are permanently kept along the map.



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "Name": "Khangchendzonga NP"
      },
      "geometry": {
        "coordinates": [
          88.31176541901834,
          27.662841119279165
        ],
        "type": "Point"
      }
    },
    {
      "type": "Feature",
      "properties": {
        "Name": "Kanchanjunga CA"
      },
      "geometry": {
        "coordinates": [
          87.92633762253854,
          27.71691739293736
        ],
        "type": "Point"
      }
    },
```

```
// Create a custom label (using divIcon)
      const label = L.divIcon({
        className: 'national-park-label',  // Custom class for styling (optional)
        html: '<strong>' + feature.properties.Name + '</strong>',  // Label text
        iconSize: [100, 30],  // Size of the label icon
        iconAnchor: [30, 0],  // Anchor the label below the marker (adjust position)
        popupAnchor: [0, -25]  // Adjust label to be positioned below the marker
      });
```

Adding APIs

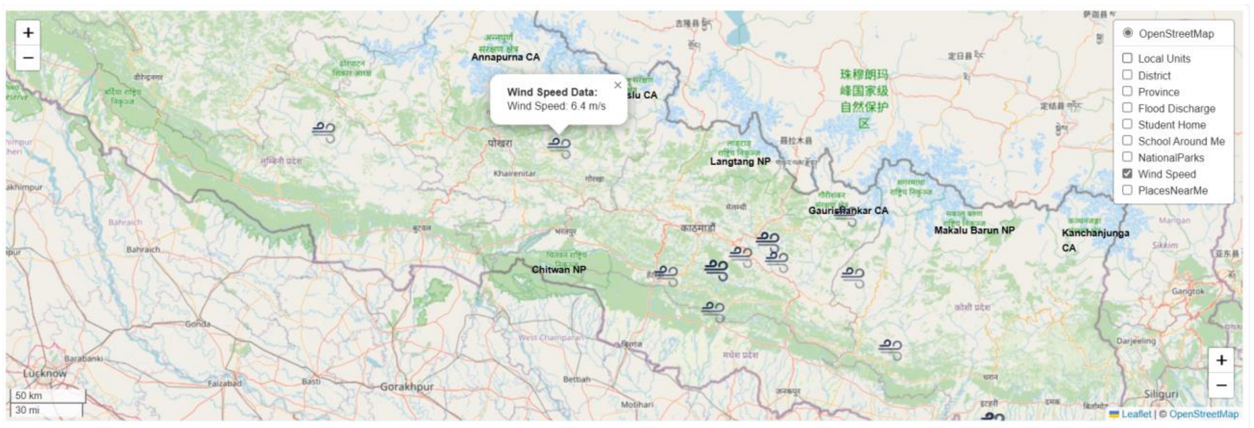
By integrating external data sources such as APIs, Leaflet maps can become dynamic platforms that display real-time data, provide detailed geographical information, and offer seamless user interactions. The integration of APIs not only improves the map's utility but also enriches the user experience by providing live, contextually relevant information. Here I have used three APIs. To fetch data from APIs we generally require **BASE URL** and **API-KEY.** For this map the APIs are fetched using Axios, which allows to fetch real-time data based on the user's click on the map. Three data are listed below with its source APIs websites

- **Flood Data: From Open-Meteo's flood API.**
- **Wind Speed: From Open-Meteo's weather API.**
- **Nearby Places: From Foursquare's API.**

I have found that only BASE URL is required to fetch the data from Open-Meteo's API and we require both URL and API-key to fetch from Foursquare's API.

The below code shows an example how we can fetch data from API (using Base URL only):

```
// Create a layer group for the flood discharge markers
    const floodLayer = L.layerGroup().addTo(map);
    function fetchFloodData(lat, lon) {
      const url = `https://flood-api.open-meteo.com/v1/flood?latitude=${lat}&longitude=${lon}&daily=river_discharge`;
      axios.get(url)
        .then(response => {const floodData = response.data.daily;
          const floodIcon = L.icon({
            iconUrl: 'myicon1.jpg',
            iconSize: [32, 32],
            iconAnchor: [16, 32],
            popupAnchor: [0, -32]
          });
          const popupContent = `
            <strong>Flood Data:</strong><br>
            River Discharge: ${floodData.river_discharge[0]} m³/s<br>
          `;
          const marker = L.marker([lat, lon], { icon: floodIcon }).addTo(floodLayer);
          marker.bindPopup(popupContent).openPopup();
        })
        .catch(error => {
          console.error('Error fetching flood data:', error);
          alert('Failed to fetch flood data.');
        });
    }
    map.on('click', function(e) {
      const lat = e.latlng.lat;
      const lon = e.latlng.lng;
      fetchFloodData(lat, lon);
    });
     const windLayer = L.layerGroup().addTo(map);
```


Wind Speed Data:
Wind Speed: 6.4 m/s

(●) OpenStreetMap
[ ] Local Units
[ ] District
[ ] Province
[ ] Flood Discharge
[ ] Student Home
[ ] School Around Me
[ ] NationalParks
[✓] Wind Speed
[ ] PlacesNearMe

Using Layer Group

a Layer Group is a powerful feature that allows us to group multiple map layers into a single entity, making it easier to manage and control them. A layer group can hold markers, polygons, lines, or even other types of layers, and it provides methods to control the visibility, adding, and removing of these layers collectively. Here we have grouped each data into the layers so we can switch on and off the layer according to our interest of visualization.

```
var baseMaps = {
        "OpenStreetMap": osm
    };

    var overlayMaps = {
        "Local Units": localunits,
        "District": districtNepal,
        "Province": provinceNepal,
        "Flood Discharge":floodLayer,
        "Student Home":studenthome,
        "School Around Me":schoolaround,
        "NationalParks":nationalpark,
        "Wind Speed":windLayer,
        "PlacesNearMe":foursquarePlacesLayer
    };
    var layerControl = L.control.layers(baseMaps, overlayMaps).addTo(map);
```