**Geography 417/817  
Assignment Three: Leaflet I**

This lab introduces how to use Leaflet layers control to allow viewers to easily **switch across different layers**on your map. This is an important part of contemporary geovisualization, as users typically expect the ability to interact with your map data rather than just viewing a static image.   
  
Unlike the previous lab, you will not be customizing a completed HTML page full of comments, rather you will be building an interactive map mostly from scratch. However, you will be following guided steps and you can scroll down further for examples of the final product. *Many parts of these instructions have been adapted from elements of Prof. Ting Liu’s GES 387 class at Northeastern Illinois University.*

This lab will focus on developing code from scratch, beginning to identify specifically what is happening at each stage. The second half of this lab will focus on styling elements.

Please read the instructions carefully (including the explanations of each step) and complete the assignment according to the “submitting your assignment” and rubric below

**Add Basemap**

Before you start, create a **new file folder** (e.g., lab2) for saving the html document and associated data file(s) to your GitHub page. Remember that you begin by adding a new file and adding a subdomain to the file location, e.g. “/lab2/index.html.” You can go back to last week’s lecture for a live demonstration, if you’ve forgotten how to add a new subfolder!

* Open your **text editor** (Notepad++ or Atom or other) and begin by adding the following lines. Note that I am **NOT defining the height of the div** in the body section and so the map will NOT yet show up at this point if you save and try to open the HTML file locally (remember from lecture that a div is the “container” for elements, in this case the map).

<!DOCTYPE html>

<html>

<head>

<title>Lab 2</title>

<link rel="stylesheet" href="https://unpkg.com/leaflet@1.3.4/dist/leaflet.css"

crossorigin=""/>

<script src="https://unpkg.com/leaflet@1.3.4/dist/leaflet.js"

crossorigin=""></script>

</head>

<body>

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

<script type="text/javascript">

var map = L.map('map', {

center: [34.666, 104.9569],

zoom: 5

});

L.tileLayer('https://stamen-tiles-{s}.a.ssl.fastly.net/toner-lite/{z}/{x}/{y}{r}.{ext}', {

attribution: 'Map tiles by <a href="http://stamen.com">Stamen Design</a>, <a href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',

subdomains: 'abcd',

minZoom: 0,

maxZoom: 20,

ext: 'png'

}).addTo(map);

</script>

</body>

</html>

* Next, in the head section, after your scripts, add the following lines to define the style of the container to make the map full screen:

<style type="text/css">

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

#map { min-height: 100%; }

</style>

We have defined the height and margins of the webpage and the div container (#map, the id of the mapped area div) for creating the fullscreen mode.  
The code is fairly intuitive, but let me know if you need more information.

* Save the document as map2.html in your assignment 2 folder – locally and (when you’re ready to your GitHub page).

**Add More Layers**

In Leaflet, there are **TWO types of layers** which allow different types of controls:  
(1) **base layers** that are generally mutually exclusive (only one can be visible at a time), *e.g. tile layers* (you could only **switch between** base layers)  
  
(2) **overlay layers**, which are all the foreground layers, *e.g., geojson*, you put on top of the base layers (you may **toggle on and off** these overlay layers)  
  
As an example, I provide code for two base layers (a light canvas and satellite imagery) to switch between, and an overlay geojson layer to turn on and off. *In the course of the assignment, you will add an additional geojson layer*!

* We will now add a new tile layer (I use satellite in the example, but you can pick anything from the leaflet extras list). However, it is a little more than just adding another L.tileLayer to make the layers control work. We actually will need to create two variables now for the two tile layers, one for each of the backgrounds, so that we could refer to them later.  
   **MODIFY** the tile layer portion to make a variable for the first map, and then add a section for this new “imagery” layer. (note: you might experience some issues if you copy+paste this):

var imagery = L.tileLayer('https://stamen-tiles-{s}.a.ssl.fastly.net/terrain/{z}/{x}/{y}{r}.{ext}', {

attribution: 'Map tiles by <a href="http://stamen.com">Stamen Design</a>, <a href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',

subdomains: 'abcd',

minZoom: 0,

maxZoom: 18,

ext: 'png'

});

var canvas = L.tileLayer('https://stamen-tiles-{s}.a.ssl.fastly.net/toner/{z}/{x}/{y}{r}.{ext}', {

attribution: 'Map tiles by <a href="http://stamen.com">Stamen Design</a>, <a href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a> &mdash; Map data &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',

subdomains: 'abcd',

minZoom: 0,

maxZoom: 20,

ext: 'png'

}).addTo(map);

* + For the light toner layer, we basically assigned the tile layer to a variable called canvas. You are free to name the variable anything else as long as it is unique. Just make sure to refer to the exact variable name later. Check out [here](https://www.w3schools.com/js/js_variables.asp) for more on JS variables.
  + And we created another tile layer using the satellite and assigned it to a variable called imagery.
  + We ONLY used the addTo(map) method for the canvas layer to use it as the **default** base map. Adding this to the map here just means it will be the “default” map that loads up.
* Next, grab the geojson layer I provided (China Provinces). This will make a very large file so we don’t want to load it directly into the HTML page. Let’s get in the habit of saving the files elsewhere and “calling them in” to the page as .JS (javascript) files as needed instead. Basically you can store any number of variables outside of the document, as long as you provide links to where that text is located.  
    
  That way, you can bring in GeoJSON files you develop with QGIS or ArcMap and use them in multiple sites. SO. Copy+paste the GeoJSON into a text editor and save it as a .JS file (e.g. provinces.js). Ensure that it is a variable, “var provinces = “ (you can give this whatever variable name you want as long as you use it the same in your HTML document) to the beginning of the file so that when you consume it into your webpage, It will know to grab this as a JS array. Upload the JS file to your GitHub. Or use [mine](https://electionmaps.github.io/provinces.js).

var provinces = {"type":"FeatureCollection","crs":{"type":"name","properties":{"name":"urn:ogc:def:crs:OGC:1.3:CRS84"}},"features":[{"type":"Feature","properties":{"GBCODE90":"340000","NAME\_PY":"Anhui","NAME\_HZ":"Anhu","POP10":59500510},"geometry":{"type":"Polygon","coordinates":[[[117.65244276239659,29.61467102623574],[117.53655289579561,29.600055666462904],[117.52491960793985,29.65990685290835],………… ; DON’T COPY THIS IT WON’T WORK

Now return to your HTML page – specifically in the head section – and add a line to call in that data file, changing my example text as needed, depending on where you uploaded it:

<script src="https://YOURNAME.github.io/lab2/provinces.js"></script>

*Or, if using mine, <script src="https://electionmaps.github.io/provinces.js"></script>*

* You will now need to add a variable to your HTML document that identifies your data source, and then have Leaflet process it as GeoJSON. SO, add a line to the body section of your HTML code within the script tags, usually easiest just to place this right after your tile layers:

var provinces = L.geoJson(provinces).addTo(map);

* Before you finish making the interactive map’s variables here, you will need to add another geojson layer. Add a data layer of early COVID cases in China. I’m hosting this on my site at:  
    
  <https://electionmaps.github.io/coviddata.js>

Use steps outlined just above (adding the script to the head and adding the variable and making sure it’s added to the map. Look at the variable name in the text of the .JS file and create a new layer in your dataset that corresponds. It’s a list of days and cases by province in China.

**Layers Control**

Once the layers are created, we set up the “control” in Leaflet. This adds options for the users to interact with your data.

* We will create two objects: one contains the base layers and one contains the overlays. I used basemaps and overlaymaps as the object names. You may use different names for the objects. Place the following lines AFTER the part of the body where you call in those variables (so, below the provinces and data layer).

var basemaps = {

"Light Canvas": canvas,

"Satellite Imagery": imagery

};

var overlaymaps = {

"China Provinces ": provinces

};

A closer look at the syntax here: these are just simple objects with key/value pairs. The key (e.g. "Light Canvas") sets the **display name** for the layer in the map control, while the corresponding value (e.g. canvas) is a **reference to the layer by variable name**.

You may style the keys/display names using HTML.

* Now, let's create a Leaflet [Layers Control](https://leafletjs.com/reference-1.3.4.html#control-layers) and add it to the map. Put this right after your overlay definitions from above:

L.control.layers(basemaps, overlaymaps, {collapsed:false}).addTo(map);

Note the L.control.layers function takes the first object as the base layers and the second object as the overlays.

Normally you will find the control button at the upper-right corner of the map area, which will allow you to switch between the basemaps, and turn on/off the overlays. Notice that I’ve set this to {collapsed:false} to ensure that it’s open by default. If you omit this section, you’ll get a button up there that users can hover over for options instead.

**Stylize Layers**

First, we are going to stylize that China provinces layer you added. You add in styles by first adding a variable to define a style. You get to choose the specific HEX color for the fill and outline of the shape. But let’s start with something really advanced:

* We’ll add two new functions to create choropleth breaks by population, for each province in China. This is a lot like creating a choropleth map in mapping software, except that nobody provides guidance on breaks.   
    
  Below is the code I used. You’re welcome to modify it. It takes values from the POP10 (population in 2010) field of the provinces’ GeoJSON file, and then groups those data into five categories represented by different colors (I sued shades of violet here). If you have changed the name of the population field from my original layer, then you need to set the attribute field accordingly. Put it right after the call to tile layers that adds them to the map. Note that you separate colors by drawing them in ascending order, with the highest amount/darkest color on top, and the bottom layer does not specify a value at all, meaning that anything that doesn’t fall into the second-lowest category (which is in this case 5 million people or more)

Copy and paste this section just after you add in the tile layers, but before you identify and call up the variables:

**function** getColor(value) {

**return** value > 50000000 ? '#54278f':

value > 25000000 ? '#756bb1':

value > 10000000 ? '#9e9ac8':

value > 5000000 ? '#cbc9e2':

'#f2f0f7';

}

**function** style(feature){

**return** {

fillColor: getColor(feature.properties.POP10),

weight: 2,

opacity: 1,

color: 'gray',

fillOpacity: 0.9

};

}

Now, modify your existing provinces variable to call up that style you created. Note that in Leaflet, you can specify properties of a L.GeoJSON layer, including symbology. In this case, we are drawing upon the style property and assigning it to a style we have defined above.

**var** provinces = new L.geoJson(provinces, **{style:style}**).addTo(map);

* Next we will be modifying the incoming infection cases / point layer so that it too has dynamic styling.

Currently, in addition to bringing in a JS file in your head section, your code calling in the infection data probably will look something like this if you did it correctly:

**var** covidData = new L.geoJson(covidData).addTo(map);

* Next, you need to modify it so that the data are scaled based on an attribute field instead of standard leaflet markers (upside down teardrop icons).
* To do this, you’ll first need to set a minimum radius and a minimum number of cases for creating circle symbols, then you will need to write a function that calculates a size of a circle based on data. You will have to create variables that begin this part by setting a minimum value and radius for the circles, then follow it up with a function that physically calculates a circle size.   
    
  Later we will use this calculation function to derive proportional symbols. Insert the following code above your point data variable, feeling welcome to change the numbers to see what works best for your dataset. *Notably, this is an iterative process, you may not immediately get circle sizes that work well for you, so adjust as needed after you complete the next step*.

**var** minValue = 100;

**var** minRadius = 5;

//the radius value is arbitrary, to produce good symbol sizes. if you're getting very large or small symbols, you can modify the math used here to determine scaling

**function** calcRadius(val) {

**return** 0.9 \* Math.pow(val/minValue, .7) \* minRadius;

}   
  
**var** coviddata = new L.geoJson(coviddata).addTo(map);

* BEFORE YOU PROCEED and check and call me over to ask why this isn’t working for you, you’ll also need to modify the point variable (cases) to allow symbol styles, too. You could do this with another variable to define styles, or you can just write it directly into the properties.   
    
  I provide an example of writing this into the properties of the L.GeoJSON layer directly below. Again, if you copy this code directly you will need to modify it based on YOUR variable names, including the variable name used in your .JS file if you changed that. And you would also need to overwrite the existing var covidData section you already began adding. Notice that the calcRadius function is fired within the radius property of the circleMarker style. I’ve left the commented part in so you could see how to set a standard circle size without scaling effects   
    
  *nb: that “ll” stands for Lat/Long – leaflet finds the location of the symbol and builds from there*

**var** covidData = new L.geoJson(covidData, **{**

**pointToLayer: *function*(feature, ll){**

**return L.circleMarker(ll, {**

**color: '#000000',**

**opacity: 1,**

**weight: 2,**

**fillColor: '#808080',**

**fillOpacity: .5,**

**//radius: 10**

**radius: calcRadius(feature.properties.covid)**

**});**

**}**

**}**)**.**addTo(map);  
  
// IF you are running into trouble here try

* Change the color of the circles to something that won’t clash with your other data, and probably something that both complements the underlying province colors AND reinforces the idea that the infections are some sort of hazard or negative thing. You can do this by modifying the fill color of the circle in the circleMarker properties.

**Map Elements**

We will need to create a variable for the legend, then add it to the map after defining its properties. This part’s a bit more complicated because we’re going to have step through colors you derived earlier based on population. I’ve provided code for this, but you will need to modify this code to match the population/color breaks you created above if you made changes. Put the legend definitions after your leaflet layer control section, but make sure it is in your BODY section, before the closing script tag.

**var** legend = L.control({position: 'bottomright'}); // Try the other lower corner if you prefer.

legend.onAdd = function (map) {

**var** div = L.DomUtil.create('div', 'legend'),

grades = [0, 1000000, 2000000, 5000000, 10000000]; // break values to define the intervals of population - YOU WILL NEED TO MODIFY IT TO GET THE RIGHT LEGEND BREAKS

div.innerHTML = '<b>Population 2019 <br><br></b>';

// The legend title, in this case it's Population 2019

// Loop through the classes or “grades” and generate a label with a colored square for each interval.

***for*** (**var** i = 0; i < grades.length; i++) {

div.innerHTML +=

'<i style="background:' + getColor(grades[i] + 1) + '"></i>' +

grades[i] + (grades[i + 1] ? '&ndash;' + grades[i + 1] + '<br><br>' : '+');

}

***return*** div;

};

legend.addTo(map);

// Removing the legend when the layer is removed

map.on('overlayremove', ***function***(eventLayer){

if (eventLayer.name == 'provinces'){ // We use the key/display name to refer to the layer

map.removeControl(legend); // legend is the variable name of the layer legend

}

});

// Adding the legend when the layer is added

map.on('overlayadd', ***function***(eventLayer){

if (eventLayer.name == 'provinces'){

legend.addTo(map);

}

});

Before running this and having it execute properly in the map, we also need to expand the CSS styles at the top of the page to *accommodate* a legend. Here is one example that gives room for the colored boxes of a choropleth legend. You can just overwrite the existing map style if you copy+paste. You should experiment with padding and placement and shadows as needed until you get the legend format you want:

#map { min-height: 100%; }

.legend {

padding: 6px 8px;

line-height: 18px;

background: rgba(255,255,255,0.9);

box-shadow: 0 0 15px rgba(0,0,0,0.2);

border-radius: 5px;

}

/\* Optional: adjust the values below to change the appearance of the legend color boxes \*/

.legend i {

width: 18px;

height: 18px;

float: left;

margin-right: 8px;

opacity: 0.7;

}

We should still have some sort of pop up here so that users can interact with the map data. We’ll work with making just one layer interactive via clicks.

* Leaflet has an option called onEachFeature. onEachFeature is a function that gets called on each feature before adding it to a GeoJSON layer. This is used most often to set the popup and define what parameters to pull out of a feature if it is clicked.
* One very organized and simple way to make this work for popup windows is just to create a variable somewhere in your script called “onEachFeature” that will define a set of options, and then just point the onEachFeature function to THAT variable, so that all those parameters are set. Let’s do this for our dataset by first creating a variable called onEachFeature with all the things we want to show in the pop up when users click it. We’ll create variables within this, too, to define the popup and properties variables.

**var** onEachFeature = ***function***(feature, layer) {

***if*** (feature.properties) {

**var** prop = feature.properties;

// make an html popup with properties

//see how you can concatenate various attributes and text as needed, //including HTML markup, with single quotes. You pull out properties by //typing in prop and then putting the field in brackets.  
//you can do this because prop = feature.properties.\_\_\_\_\_

**var** popup = '<h3>'+prop['Location']+'</h3>'+'<br>Cases Day 1: '+prop['1']+'<br>Cases Day 16: '+prop['covid'];

// add known info about event to the description

// you must create a layer property on each feature or else

// the search results wont know where the item is on the map / layer

feature.layer = layer;

layer.bindPopup(popup, {maxWidth: "auto"});

}

}; // end onEachFeature

Set the pop up to reflect fields that you used in your dataset. Customize however you like, but it should at least show the location and the number of cases on some day (probably the day you’re using for the circle values). Then, you will also need to fire the onEachFeature on the layer.

* Here’s where you fit this into your layer call after you’ve defined the variable:

**var** coviddata = new L.geoJson(covidData, {

**onEachFeature : onEachFeature,**

pointToLayer : ***function***(feature, ll){

return L.circleMarker(ll, {

color: '#000000',

opacity: 1,

weight: 2,

fillColor: '#808080',

fillOpacity: .5,

//radius: 10

radius: calcRadius(feature.properties.covid)

});

}

**Submitting Your Assignment**

* Your final map will be graded based on its **overall accuracy (should meet all the requirements above)** and **cartographic layout/design**.
* Host your web page via GitHub (Make sure to rename the html file as index.html and upload any associated data files to the repository!!!) and submit the url through Canvas!
* In that word document, answer the following questions (5pts each)

1) How would you add *another* *class* to your choropleth map?

2) How would you add another class to your *legend*?

3) How might you create a map where all map layers are toggled OFF by default? Why might this be a useful option?