# Practical Exercise 3: Thematic Web Map Design

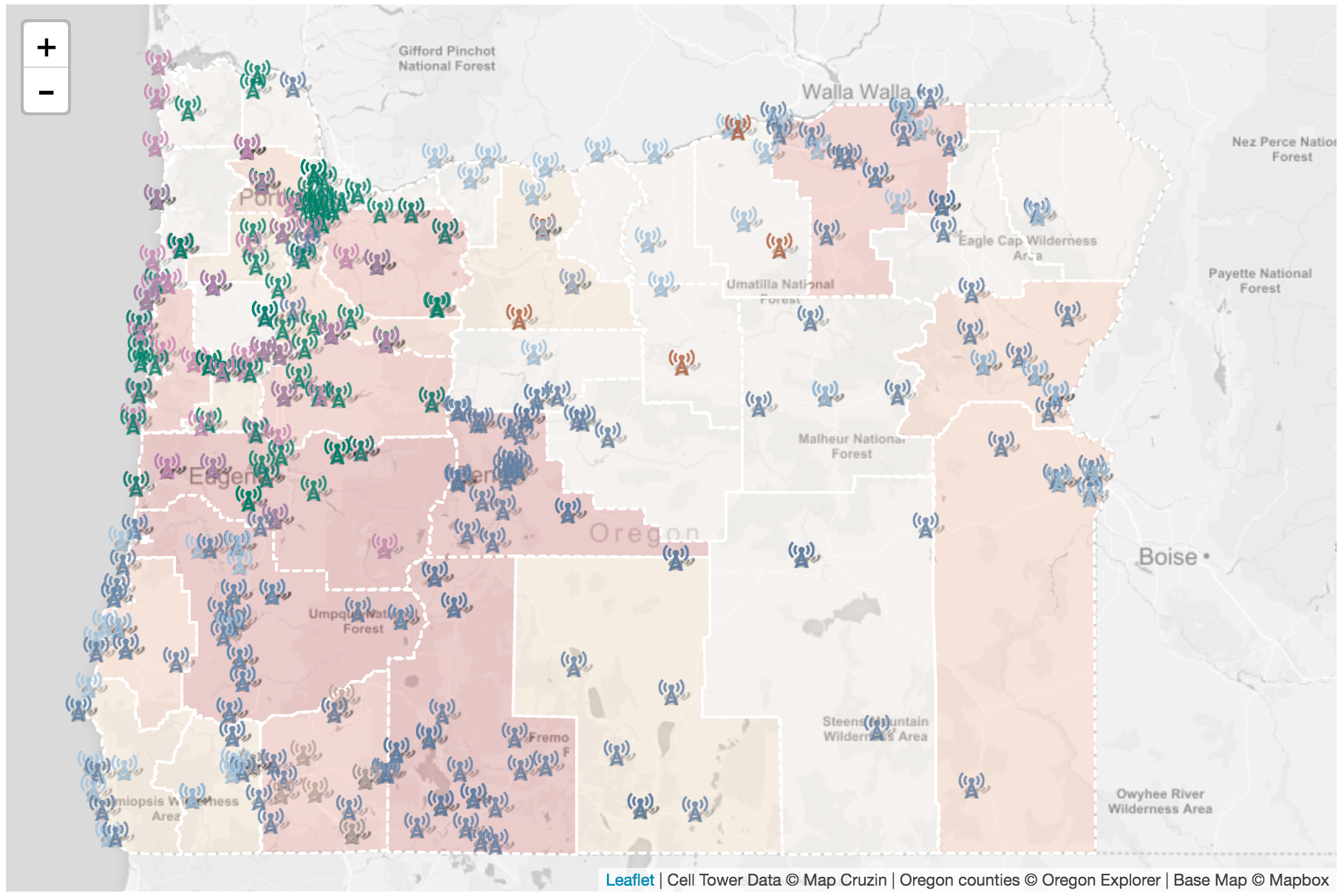
Winter 2017 | Geography 371 | Geovisualization: Web Mapping

Instructor: Bo Zhao | TA: Andy Wilson | Location: 210 Wilkinson | Time: Thursday 2-3:50pm

Assigned: 01/26/2017 | Due: 02/09/2017 @11:59pm | Points Available = 50

It's time to look at some cartographic components of our map, including map elements, symbolization, and customization. When creating a web map, one of the key components is styling your elements to provide proper symbolization for your data. This increases legibility for users and can give your map an appealing, custom design. Elements that can be custom designed include thematic layers (points, lines, and polygons), basemaps (tile layers), interactive features (the components of the map that allow for user interaction), and legends and supplemental information (such as supplemental prose and titles).

This lab, let us make a web map of cell towers in Oregon. To do that, we collected all the county boundaries from [Oregon Explorer](http://oregonexplorer.info), and the national distribution of cell towers from [Map Cruzin](http://www.mapcruzin.com/google-earth-maps-resources/kml/us-cell.kmz). To get a visual, this is what we are going to make today.



To get started, setup your development environment in a easy to access location on your machine. Download this lecture's package from Canvas, unzip the contents, and start up your localhost server in that location (the development environment). Again, you can start up your local server by python SimpleHTTPServer or Webstorm.

## 1. Set up our Map and Add Data

With your localhost running, open up the **map1.html** in your IDE (Webstorm) to prepare for editing. When open, you should see a basic map showing the extent of Oregon State.

**The extent of Oregon State**

View this Example View the map1.html document in Webstrom (or your text editor of choice). Here you will see our HTML, with some CSS styling at the top, a couple of div page elements for our map components, linked scripts, and custom scripts. The page elements are as follows:

wrapper: the main container for our map in the body. Our whole map and interface will fall inside the wrapper element;

map: the element our map will be attached to;

controls: the element in which we can put any buttons or controls;

credits: an element inside controls where we can put our contact and copyright information.

**The Script**

Within the script tags, I have added the map object and tile layer for us to use. You've seen the script before. This script creates our map object and adds a basemap.

// Create Map Object  
var mymap = L.map('map', {center: [44.13, -119.93], zoom: 7});  
  
// Add Base Layer  
L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access\_token=pk.eyJ1IjoibWFwYm94IiwiYSI6ImNpandmbXliNDBjZWd2M2x6bDk3c2ZtOTkifQ.\_QA7i5Mpkd\_m30IGElHziw', {  
 maxZoom: 11,  
 minZoom: 6,  
 attribution: 'Cell Tower Data &copy; Map Cruzin | Oregon counties &copy; Oregon Explorer | Base Map &copy; Mapbox',  
 id: 'mapbox.light'  
}).addTo(mymap);

Please visit <http://localhost:8000/map1.html> to see the map at the current stage.



The tile layer I am using comes from Map Box. The light color helps stand out the main features/theme of a web map. In addition to worldwide base layers you find from major organizations (Google map, bing map, openstreetmap), many local municipalities and regions around the globe maintain tile layers that can be accessed through GIS software and mapping libraries. Google search the area in which you are working to see if they maintain base maps, most often you will find Web Map Services (WMS) - a map services we will learn in later, that can be loaded into your map as a base using the Leaflet WMS loader object, L.tileLayer.WMS.

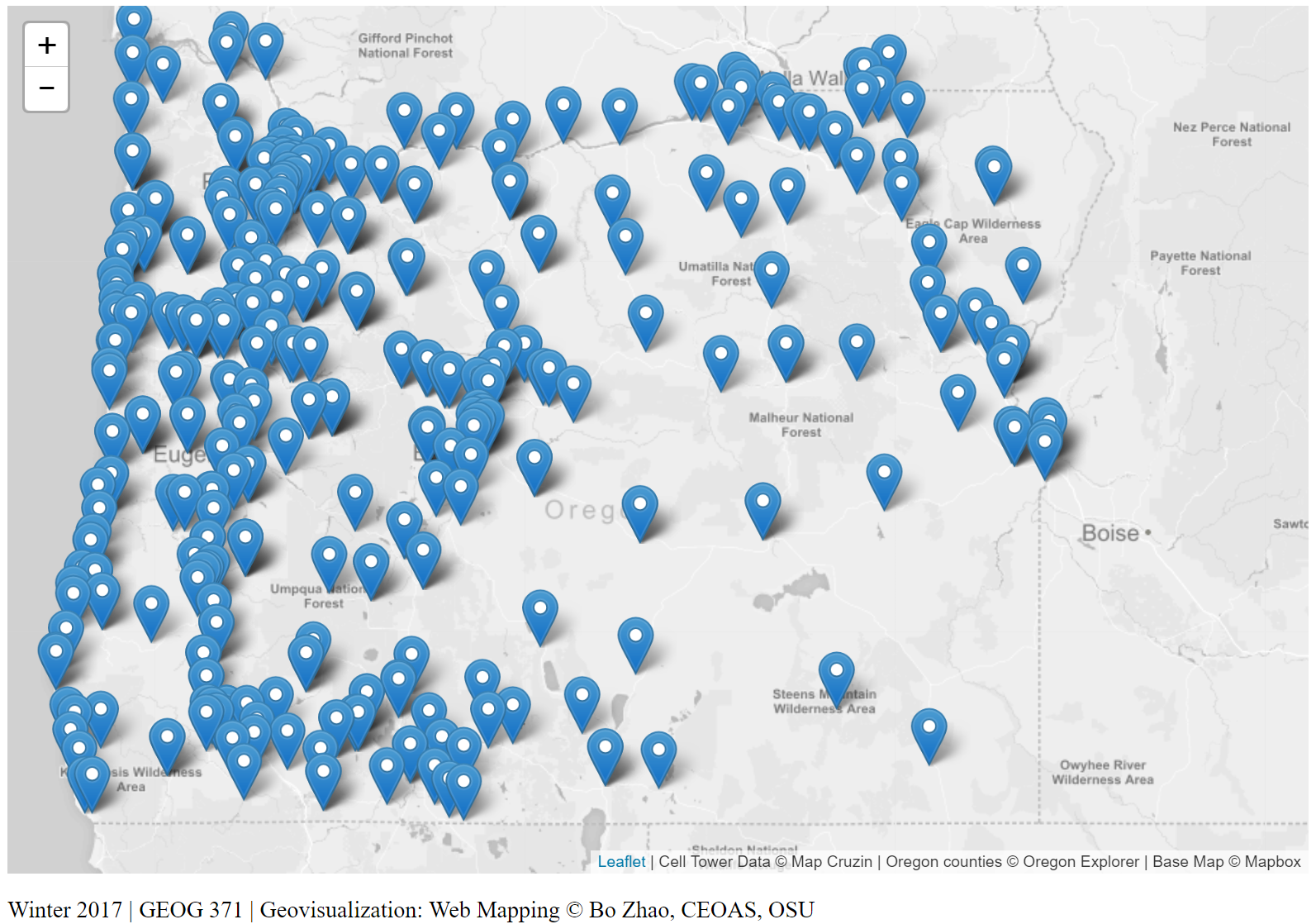
**Note:** If the provided basemaps do not carter to your flavor, you can create custom tiles that can be served to your maps. This topic, on the whole, is large and we will have another series of lectures that introduces creating web map services by GeoServer.

**Add the Cell Towers Data**

Next, we want to add the dataset to the map. Note that I have included the jQuery library in our document so we can use an Asynchronous call to get external data (AJAX) and the $.getJSON method to add our dataset from an external location. In the data folder of the downloaded materials, you will find our dataset, cell\_towers.geojson. At the end of our script, within the script tags, enter the following code to add the cell towers dataset and bind a popup.

// Add cell towers GeoJSON Data  
// Null variable that will hold rodent violation data  
var cellTowers = null;  
  
// Get GeoJSON and put on it on the map when it loads  
$.getJSON("assets/cell\_towers.geojson",function(data){  
 // set cellTowers to the dataset, and add the cell towers GeoJSON layer to the map  
 cellTowers = L.geoJson(data,{  
 onEachFeature: function (feature, layer) {  
 layer.bindPopup(feature.properties.company);  
 }  
 }).addTo(mymap);  
});

The cellTowers variable will hold the contents of our GeoJSON, so we can refer to it easily. Save and refresh your map. You should see the points populate. That is a lot of cell towers! Please open **map2.html** to see the map at the current stage.



## 2. Custom Point Markers

Our point markers showing the cell towers are the default blue Leaflet map pin. While these markers are fine, if you are showing multiple properties or want to create unique symbols, you can set your point symbols to be represented by an icon of your choosing. The steps towards doing this are quite easy, and you can use the Leaflet icon class to set up your parameters. You have two choices for your custom icons. First, you can use existing icons or grab a library of pre made icons. Here are two nice leaflet plugins:

* Mapbox Maki: allow you to use icons from the terrific Mapbox Maki library.
* Font Awesome: Allows for use of the fantastically useful open source.

Second, you can make your own icons by using an existing image or creating one using graphics software (i.e. Illustrator, Photoshop). If your graphic is saved as an image (the most space efficient images for the web are usually in png or jpg format), and upload it to your server for use as an icon. If you want a higher level of customization, or the icons found in **Font Awesome** or **Maki** do not work for you, create your own!

### 2.1 Create a Custom Marker

In the downloaded data package for this week, you will see a folder named img. It contains a handful of images that can be used as markers. To get started, since we are working with rodent incidents, lets change our default blue map pin to be an icon of a cell tower: (ct.png).

*Use the L.icon object and create a holder for our custom cell tower icon*

The first task is to create an object that will hold our custom icon. Enter the following code to created our custom rodent icon, within the script tags. Note it will be a global variable, meaning we can use the variable at any location in our script.

// Create Custom Icons Here  
var TowerIcon = L.Icon.extend({  
 options:{  
 iconUrl: 'img/ct.png',  
 shadowUrl: 'img/ct\_sd.png',  
 iconSize: [18, 18],  
 shadowSize: [25, 18],  
 iconAnchor: [16, 16],  
 shadowAnchor: [16, 16],  
 popupAnchor: [-8, -18]  
 }  
});

This used the Leaflet Icon class (L.icon) to set up the icon. L.icon requires a path to your icon, then takes a handful of other options that allow you a high level of control. Creating this icon as an object and saving it as TowerIcon will allow us to set the display of feature to this icon. Let's talk a bit more about these options.

* iconUrl: contains the path to your icon
* shadowUrl: contains the path to the icon shadow to give a 3D feel to your map
* iconSize: sets the size of your icon in pixels. Best to work at size, okay to scale down, never scale up
* shadowSize: sets icon shadow image size
* iconAnchor: sets anchor point (where the icon is located in respect to the feature latitude and longitude
* shadowAnchor: sets anchor point of shadow
* popupAnchor: sets anchor point of bottom of popup

*Use point to layer option of L.geoJson to set the icon*

With our icon loaded into our document, we need to replace the default icons created when we add the GeoJSON. This is a process of setting the icon option to TowerIcon for each marker when it is added to our map. To set the icon for a GeoJSON, we need to create a layer from the GeoJSON (we can style it if it is a layer) by using the pointToLayer option of L.geoJson. Our GeoJSON is added to our map using jQuery with the following block of code.

// Get GeoJSON and put on it on the map when it loads  
$.getJSON("assets/cell\_towers.geojson",function(data){  
 // set cellTowers to the dataset, and add the cell tower GeoJSON layer to the map  
 cellTowers = L.geoJson(data,{  
 onEachFeature: function (feature, layer) {  
 layer.bindPopup(feature.properties.company);  
 }  
 }).addTo(map);  
});

**Options available for L.geoJson include**:

* pointToLayer: Function that will be used for creating layers for GeoJSON points (if not specified, simple markers will be created).
* style: Function that will be used to get style options for vector layers created for GeoJSON features.
* onEachFeature: Function that will be called on each created feature layer. Useful for attaching events and popups to features.
* filter: Function that will be used to decide whether to show a feature or not.
* coordsToLatLng: Function that will be used for converting GeoJSON coordinates to LatLng points (if not specified, coords will be assumed to be WGS84 — standard [longitude, latitude] values in degrees).

We are using onEachFeature to set the popup, but you can see that in order to set a icon, we need to use pointToLayer. In pseudo-code, pointToLayer runs a function when the GeoJSON is loaded that takes a feature and latitude and longitude and creates a marker at that latitude and longitude. Marker has an option called icon that you set to be our towerIcon variable. Once set, return the marker. This will replace the default blue map pin with our rodentIcon. The code looks like the following. Replace your L.geoJson call with this. Note the addition of the pointToLayer option.

// Get GeoJSON and put on it on the map when it loads  
$.getJSON("assets/cell\_towers.geojson",function(data){  
 // set cellTowers to the dataset, and add the cell tower GeoJSON layer to the map  
 cellTowers = L.geoJson(data,{  
 onEachFeature: function (feature, layer) {  
 layer.bindPopup(feature.properties.company);  
 }, pointToLayer: function (feature, latlng) {  
 var marker = L.marker(latlng,{icon: towerIcon});  
 return marker;  
 }  
 }).addTo(map);  
});

Click save and refresh your map in your browser. Check out our map. We have changed the icon to a cell tower!

### 2.2 Create an Icon Class and Change Icons for cell towers from differnet companies

If we want to symbolize which wireless company the cell tower belongs to, we can create an icon class. One of the goals of programming is to **never repeat code**, and creating a class for the icons will allow us specify anchors and sizes once, and we can change out icons. In our GeoJSON, the status property contains this information for each feature and can be accessed by feature.properties.status. We probably don't want to enter all of the specifics on sizes and anchors again, they will remain the same. We can create a class of icons that will allow us to only specify our icon image, keep all other options unchanged.

**Icon Class**

// Create Custom Icons Here  
var TowerIcon = L.Icon.extend({  
 options:{  
 shadowUrl: 'img/ct\_sd.png',  
 iconSize: [18,18],  
 shadowSize: [25,18],  
 iconAnchor: [16, 16],  
 shadowAnchor: [16, 16],  
 popupAnchor: [-8, -18]  
 }  
});  
  
var tower1Icon = new TowerIcon({iconUrl: 'img/ct1.png'});  
var tower2Icon = new TowerIcon({iconUrl: 'img/ct2.png'});  
var tower3Icon = new TowerIcon({iconUrl: 'img/ct3.png'});  
var tower4Icon = new TowerIcon({iconUrl: 'img/ct4.png'});  
var tower5Icon = new TowerIcon({iconUrl: 'img/ct5.png'});  
var tower6Icon = new TowerIcon({iconUrl: 'img/ct6.png'});  
var tower7Icon = new TowerIcon({iconUrl: 'img/ct7.png'});  
var tower8Icon = new TowerIcon({iconUrl: 'img/ct8.png'});  
var tower9Icon = new TowerIcon({iconUrl: 'img/ct9.png'});

The class option is built on top of, or extends, the L.icon object. More reading can be found on defining icon classes in the Leaflet documentation. An important note, classes begin with a capital letter (Icon versus icon).

**Use a Conditional in our get GeoJSON to determine status**

We have in our document nine icons for differnet wireless companies, such as New Cingular, Verizon, Cello, Salem Cellular, etc. When the GeoJSON is added to the map, we need to check the features when we apply the custom icon to see what the value of feature.property.company is. If it is equal to "Verizon", we want the icon to be set to one of the towerXIcon (X = 1, 2, 3..9). Here we learned about conditionals, specifically If.. Else statements. To accomplish this, we can put a conditional in our call to the GeoJSON that checks to see if a case status is equal to "Verizon" and then sets an icon, and if it is not, will run the else statement, setting the icon equal to other companies, and so on so forth. The code will look like this:

if (feature.properties.company == "New Cingular"){  
 marker = L.marker(latlng,{icon: tower1Icon});  
} else if (feature.properties.company == "Cello"){  
 marker = L.marker(latlng,{icon: tower2Icon});  
} else if (feature.properties.company == "Hood River Cellular"){  
 marker = L.marker(latlng,{icon: tower3Icon});  
} else if (feature.properties.company == "Medford Cellular"){  
 marker = L.marker(latlng,{icon: tower4Icon});  
} else if (feature.properties.company == "Verizon"){  
 marker = L.marker(latlng,{icon: tower5Icon});  
} else if (feature.properties.company == "Oregon RSA"){  
 marker = L.marker(latlng,{icon: tower6Icon});  
} else if (feature.properties.company == "RSS Minnesota"){  
 marker = L.marker(latlng,{icon: tower7Icon});  
} else if (feature.properties.company == "Salem Cellular"){  
 marker = L.marker(latlng,{icon: tower8Icon});  
} else {  
 marker = L.marker(latlng,{icon: tower9Icon});  
}  
return marker;

Note there are *two equal signs (==)*, this is because JavaScript is very particular about operators. To read more, check out this documentation from w3schools. Modify the code within the L.geoJSON pointToLayer option, where we set the style previously, to be the following, including the conditional statement to check the status.

// Get GeoJSON and put on it on the map when it loads  
$.getJSON("assets/cell\_towers.geojson", function(data){  
 // add the cell tower GeoJSON layer to the map  
 cellTowers = L.geoJson(data,{  
 onEachFeature: function (feature, layer) {  
 layer.bindPopup(feature.properties.company);  
 }, pointToLayer: function (feature, latlng) {  
 var marker = null;  
 if (feature.properties.company == "New Cingular"){  
 marker = L.marker(latlng,{icon: tower1Icon});  
 } else if (feature.properties.company == "Cello"){  
 marker = L.marker(latlng,{icon: tower2Icon});  
 } else if (feature.properties.company == "Hood River Cellular"){  
 marker = L.marker(latlng,{icon: tower3Icon});  
 } else if (feature.properties.company == "Medford Cellular"){  
 marker = L.marker(latlng,{icon: tower4Icon});  
 } else if (feature.properties.company == "Verizon"){  
 marker = L.marker(latlng,{icon: tower5Icon});  
 } else if (feature.properties.company == "Oregon RSA"){  
 marker = L.marker(latlng,{icon: tower6Icon});  
 } else if (feature.properties.company == "RSS Minnesota"){  
 marker = L.marker(latlng,{icon: tower7Icon});  
 } else if (feature.properties.company == "Salem Cellular"){  
 marker = L.marker(latlng,{icon: tower8Icon});  
 } else {  
 marker = L.marker(latlng,{icon: tower9Icon});  
 }  
 return marker;  
 }  
 }).addTo(mymap);  
});

Please visit <http://localhost:8000/map3.html> to see the map at the current stage.

## 3. Polygon Data and Symbolization

In our data folder, you'll see a second dataset, counties.geojson. The counties GeoJSON file contains all the counties of Oregon. Each county contains the number of cell towers. This number is pre-calculated in QGIS. To add the data to the map, create another L.geoJson object using the jQuery $.getJSON method. Enter the following lines of code at the end of your script, staying within the script tags.

// Null variable that will hold neighborhoods layer  
var neighborhoodsLayer = null;  
  
// Add Neighborhood Polygons  
$.getJSON("data/cambridge\_neighborhoods.geojson",function(data){  
 neighborhoodsLayer = L.geoJson(data).addTo(map);  
});

Save and refresh your map. Counties of Oregon will be displayed on the map, symbolized in a default blue.

On its own (take a look at Page Source) Let's do something about that default blue and thematically style our data to these polygons useful by turning them into a choropleth layer. The neighborhoods GeoJSON file contains numbers of cell towers in each counties, calculated in QGIS. Symbolize the counties on the map by the number of counties. This is a three step process. L.geoJSON contains a style option that contains styling properties.

### 3.1 Set up function for Color Ramp

The first step is to set up a function for our color ramp. This is where we set up our classification breaks and color scheme. Setting up a classification scheme can be tricky. The easiest way to set up a standard classification scheme is to use QGIS, select something like Jenk's Natural Breaks, and copy the break numbers. Or you can check out a color ramp from colorbrewer2.org To set up the function for our classes and colors, create a function, call it setColor, and then return your classes. Enter the following function in your script tags.

// Set function for color ramp  
function setColor(density){  
 return density > 18 ? '#b30000' :  
 density > 13 ? '#e34a33' :  
 density > 10 ? '#fc8d59' :  
 density > 5 ? '#fdcc8a' :  
 '#fef0d9';  
}

### 3.2 Set style GeoJSON style function

Next, set up a function that will set the properties for the L.geoJson style option. Call this function style, pass it a GeoJson feature when invoked. Set the fillColor property to be equal to the setColor function, passing it the CN\_CNT property (the number of cell towers) from the GeoJSON. Enter the following code into your script.

// Set style function that sets fill color property equal to cell tower density  
function style(feature) {  
 return {  
 fillColor: setColor(feature.properties.CT\_CNT),  
 fillOpacity: 0.2,  
 weight: 2,  
 opacity: 1,  
 color: '#ffffff',  
 dashArray: '4'  
 };  
}

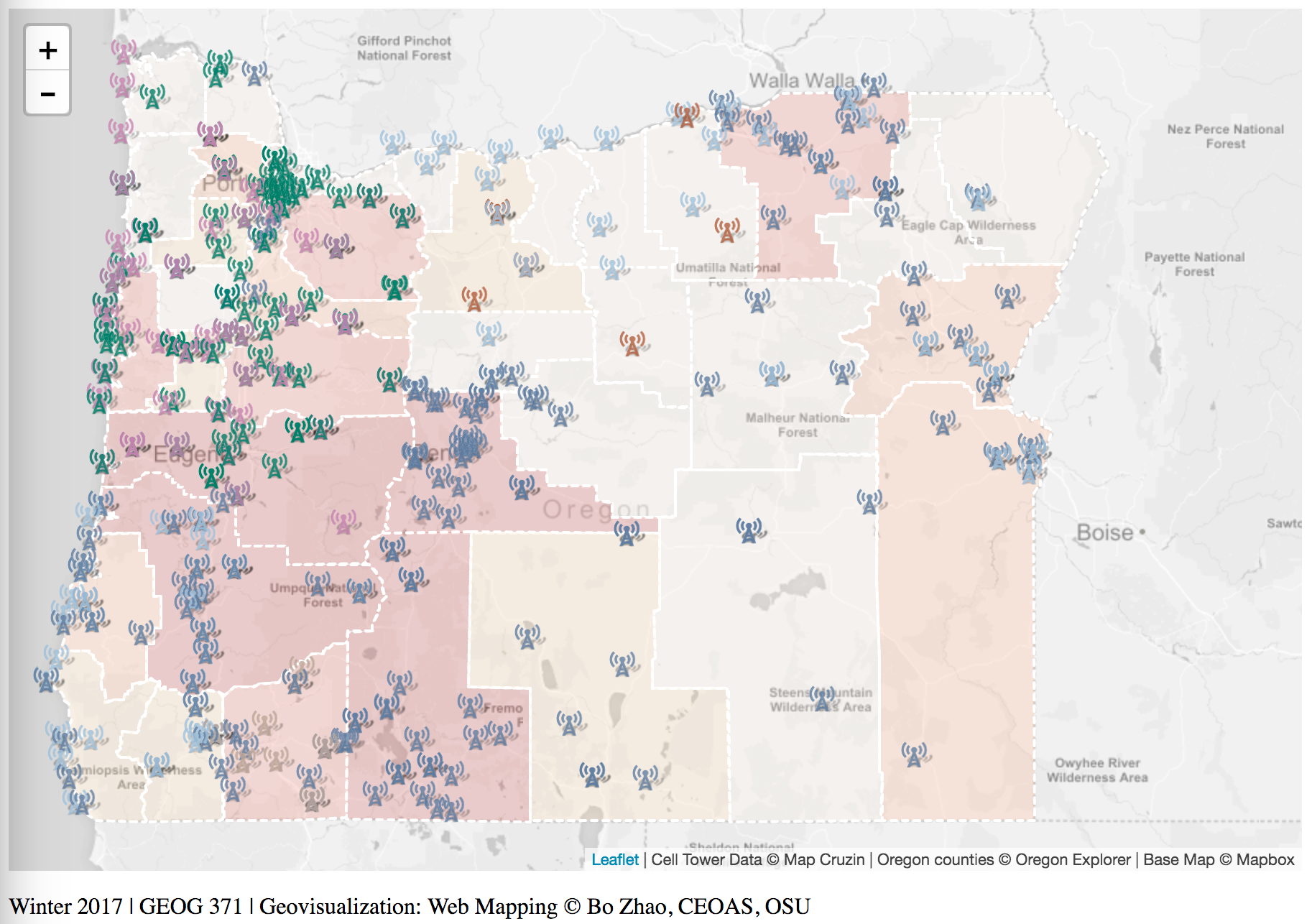
fillColor and fillOpacity set properties for the fill, and weight, opacity, color, and dashArray set properties for the border.

### 3.3 Set style option for the GeoJSON

The final step is to set the style option for the neighborhoods layer GeoJSON. Find the Add Neighborhoods Polygons block of code and modify it to include the style option. Set style equal to style to run the style function when the GeoJSON is create. Below shows the code of adding the county polygons to the map.

// Add Counties Polygons  
$.getJSON("assets/counties.geojson",function(data){  
 countiesLayer = L.geoJson(data, {style: style}).addTo(mymap);  
});

Save and refresh your map. Load your page to see our styled polygons!



## 4. Map Elements

Our map is looking good, but we need a legend to make sense of our data. We could enable popups for each of the counties, but with popups already enabled on our points, it might be overwhelming to the user. Instead, let us add a legend to our map that contains the information the reader will need to know about the data, colors, and classifications, and then add a scale bar to the corner of the map. The main Leaflet object we will use in this section is the Control object, or L.control. It allows for adding various elements to your map.

### 4.1 Add a Legend

Adding a legend is easy, but requires quite a bit of code. The workflow to create a legend involves creating a Leaflet control, setting the control to populate with HTML that represents the legend components, and styling the HTML with CSS so they appear properly on our screen. I'm going to throw a bit more code at you this time, and we will walk through what it is doing. Enter the following block of code to your script (stay in those script tags!).

// Create Leaflet Control Object for Legend  
var legend = L.control({position: 'topright'});  
  
// Function that runs when legend is added to map  
legend.onAdd = function () {  
  
 // Create Div Element and Populate it with HTML  
 var div = L.DomUtil.create('div', 'legend');  
 div.innerHTML += '<b># Cell Tower</b><br />';  
 div.innerHTML += '<i style="background: #b30000; opacity: 0.5"></i><p>19+</p>';  
 div.innerHTML += '<i style="background: #e34a33; opacity: 0.5"></i><p>14-18</p>';  
 div.innerHTML += '<i style="background: #fc8d59; opacity: 0.5"></i><p>11-13</p>';  
 div.innerHTML += '<i style="background: #fdcc8a; opacity: 0.5"></i><p> 6-10</p>';  
 div.innerHTML += '<i style="background: #fef0d9; opacity: 0.5"></i><p> 0- 5</p>';  
 div.innerHTML += '<hr><b>Company<b><br />';  
 div.innerHTML += '<img src="img/ct1.png"><p> New Cingular</p>';  
 div.innerHTML += '<img src="img/ct2.png"><p> Cello</p>';  
 div.innerHTML += '<img src="img/ct3.png"><p> Hood River Cellular</p>';  
 div.innerHTML += '<img src="img/ct4.png"><p> Medford Cellular</p>';  
 div.innerHTML += '<img src="img/ct5.png"><p> Verizon</p>';  
 div.innerHTML += '<img src="img/ct6.png"><p> Oregon RSA</p>';  
 div.innerHTML += '<img src="img/ct7.png"><p> RSS MInnesota</p>';  
 div.innerHTML += '<img src="img/ct8.png"><p> Salem Cellular</p>';  
 div.innerHTML += '<img src="img/ct9.png"><p> US Cellular</p>';  
 // Return the Legend div containing the HTML content  
 return div;  
};  
  
// Add Legend to Map  
legend.addTo(mymap);

So, what did we do here? First, we created an instance of a **Leaflet Control object**, calling it Legend, and used the position option to tell it to locate in the top right of our map. Next, we used the onAdd method of the control to run a function when the legend is added. That function creates a new div in the DOM, giving it a class of legend. This will let us use CSS to style everything using the legend tag. In the newly created div, we are going to populate it with HTML by using a built-in JavaScript DOM method called innerHTML. Using innerHTML allows us to change the content of the HTML and add to the legend div. Using the plus-equal += instead of equal = is the syntax for append. Everytime this is used, code following is appended to existing code. In this, we write the HTML we want to use in our legend. Note, the i tag represents our legend icons. Within the HTML, fill in the colors and ranges so that they match our data classfication. After the HTML is appended, return the div element. Lastly, add the legend to the map.

**Use CSS to Style**

If we save and refresh, the items you see won't make much sense, we need to use CSS to give them placement and organization on the page. The following CSS code will style our elements. Enter it between the style tags in the head of your HTML document. Like above, we'll then walk through what it does.

.legend {  
 line-height: 16px;  
 width: 130px;  
 color: #333333;  
 font-family: 'Open Sans', Helvetica, sans-serif;  
 padding: 6px 8px;  
 background: white;  
 background: rgba(255,255,255,0.5);  
 box-shadow: 0 0 15px rgba(0,0,0,0.2);  
 border-radius: 5px;  
}  
  
.legend i {  
 width: 16px;  
 height: 16px;  
 float: left;  
 margin-right: 8px;  
 opacity: 0.9;  
}  
  
.legend img {  
 width: 16px;  
 height: 16px;  
 margin-right: 3px;  
 float: left;  
}  
  
.legend p {  
 font-size: 12px;  
 line-height: 16px;  
 margin: 0;  
}

First, we set properties for the legend on the whole using .legend to style the legend class. We set a line height, color, font, padding, background, drop shadow, and border corner radius. Next we set our icon (i) tag, this should be set to float: left; so that elements will align into columns, then we set properties for the image (img) tag, making them the same size and giving them the same float as the icons. Lastly, we style our paragraph tag (p), making sure line-height is consistent with the others. Save and refresh your map. You should see your styled legend applied to your map.

This is not the only way to create a legend. An alternative method uses a For loop conditional statement to add legend elements based on the number of data classification categories you have. A [legend section of the choropleth tutorial](http://leafletjs.com/examples/choropleth.html#custom-legend-control) at the Leaflet homepage shows this, take a look.

### 4.2 Add a Scale Bar

The Leaflet Control object allows you to add a number of elements, including attribution and zoom controls. One easy component to add is a scale bar. In your script, enter the following line to add a scale bar to your map.

// Add Scale Bar to Map  
L.control.scale({position: 'bottomleft'}).addTo(map);

Save and refresh. The current state of our map.

## 5. Style your Interface

Let's finish today with some interface customization. Let's just do two simple things to further customize our interface, change the font, and right justify the credits.

**Change the Fonts**

Choosing fonts is an important part of cartography, and an often overlooked one. Right now, our map uses the default Browser font, usually Times New Roman. To edit fonts, we want to style CSS. In CSS, there are a lot of options for fonts, for more reading, check out the [w3schools font documentation](http://www.w3schools.com/css/css_font.asp).

Traditionally, a font is loaded into your page only if you have it on your computer. This presents a problem though, if someone doesn't have the font, it will change the page to use secondary or default fonts. In order to ensure that every visitors computer display the same, you can link to online font libraries. A common, useful online font library is Google Fonts. Google fonts can be added to any site, and since you link to the style, you don't have to worry about the user not having the font installed on their computer. Check out the Google Font library and explore their options. Let's link a common web font called Open Sans to our document so we can use it. To link it to our document, enter the following line of code into the head section of your document. It should go right after your stylesheets.

<head>...  
<link href='http://fonts.googleapis.com/css?family=Open+Sans' rel='stylesheet' type='text/css'>  
...</head>

Next, to style all text in our document with the Open Sans font, modify the body tag in the CSS (the code between the style tags). Modify the body CSS properties to look like the following, adding a font-family property after margin.

body {  
 margin: 0px;  
 font-family: 'Open Sans', Helvetica, sans-serif;  
}

Save and refresh your map. Open Sans will now be your preferred font!

**Style the Credits using CSS**

Lastly, to help you explore the power of CSS, style the credits at the bottom of your page. Because the div containing the credits has id="credits", we can style it using #credits. All of the contents in the credits div are between to paragraph tags. CSS styling is written in a nested fashion, to style everything that is in a p element within the #credits div, we use #credits p. Add the following snippet between the style tags in the head section of the document.

credits p {  
 margin-top: 5px;  
 font-size: 12px;  
 text-align: right;  
 line-height: 16px;  
}

Save and refresh your map. It will look like the following!

Challenge!! Can you add some interactivity by using JavaScript to implement buttons, checkboxes, and toggles?

## 6. Deliverable

Please walk through the web map of cell towers in Oregon (2009) step by step. After you successfully deploy the cell tower map, this practical exercise asks you build another web map of airports in United States. In the package for this lab, you will see two geojson files in the assets folder , one is airports.geojson, another is us-states.geojson. In addition, the img folder contains three icons may be useful, an airport icon airport1.png in black, an airport icon airport2.png in red, and an airport shadow icon airport\_sd.png.

airports.geojson is a geojson data file containing all the airports in United States. This data is converted from a shapefile, which was downloaded and unzipped from <https://catalog.data.gov/dataset/usgs-small-scale-dataset-airports-of-the-united-states-201207-shapefile.> For each airport feature, the field CNTL\_TW indicates whether the airport has an air traffic control tower. If there is a tower, the value of CNTL\_TW is 'Y', otherwise 'N'. Please use the two different airport icons (e.g., airport1.png and airport2.png, this two airport icons is converted from the airport svg file from <https://commons.wikimedia.org/wiki/File:Plane_icon_nose_up.svg>) to visualize whether an arbitrary airport has a control tower or not. **(10 points)**

us-states.geojson is a geojson data file containing all the states boundaries of United States. This data is acquired from from [Mike Bostock](http://bost.ocks.org/mike) of [D3](http://d3js.org/). The count field indicates the number of airports within the boundary of the state under investigation. So please make a choropleth map based on the count of airports within each state. **(10 points)**

Regarding the web map of airports in United States, we would like to see (part of the grading criteria)

* an appropriate basemap; **(5 points)**
* some interactive elements, like a clickable marker; **(5points)**
* the fundamental web map elements, such as legend, scalar bar, credit; **(5 points)**
* add on a new feature (e.g., a map control? a map event, a new map object, etc.) which is not introduced in this lab as well as other lectures of the web map client series; **(10 points)**
* wrap up a few words on the new feature. These words, coupled with a short introduction to this web map. These text should be added to the web map page as a short paragraph within the <p> tag. **(5points)**

Once you finish this PE, please make sure the program is executable in a web environment (testing it with python SimpleHTTPServer or Webstorm), and then compressed the folder of all the files ( web page, icon images, and javascript scripts) as a zip file package.zip to **Canvas Dropbox**. The internal structure of the zipped package should be something looks like below. Please contact the instructor or TA if you have any difficulty in compressing the files.

PE3\_.zip  
│index.html  
├─assets  
│ airports.geojson  
│ us-states.geojson  
├─css  
│ main.css  
├─img  
│ airport1.png  
│ airport2.png  
│ airport\_sd.png  
└─js  
 main.js

On the assignment tab, check the item of this PE, press the Submit Assignment button to submit your PE zipped program. Note: only submit your PE report in Zip format.

## Reference

[1] <http://duspviz.mit.edu/web-map-workshop/map-symbolization/>

[2] Data source: <http://www.mapcruzin.com/google-earth-maps-resources/google-earth-cell-towers.htm>

[3] Boundary: <http://oregonexplorer.info/ExternalContent/SpatialDataForDownload/RCE_counties.zip>

[4] open source icon <http://ionicons.com/>

[5] Cell Tower Icon <http://www.freeiconspng.com/free-images/tower-icon-25704>

[6] Marker Shadow Maker <https://dominoc925-pages.appspot.com/webapp/create_icon_shadow/default.html>

[7] Marker color <http://www.pantone.com/fashion-color-report-fall-2016#intro>

[8] Add topojson instead of geojson <http://blog.webkid.io/maps-with-leaflet-and-topojson/>