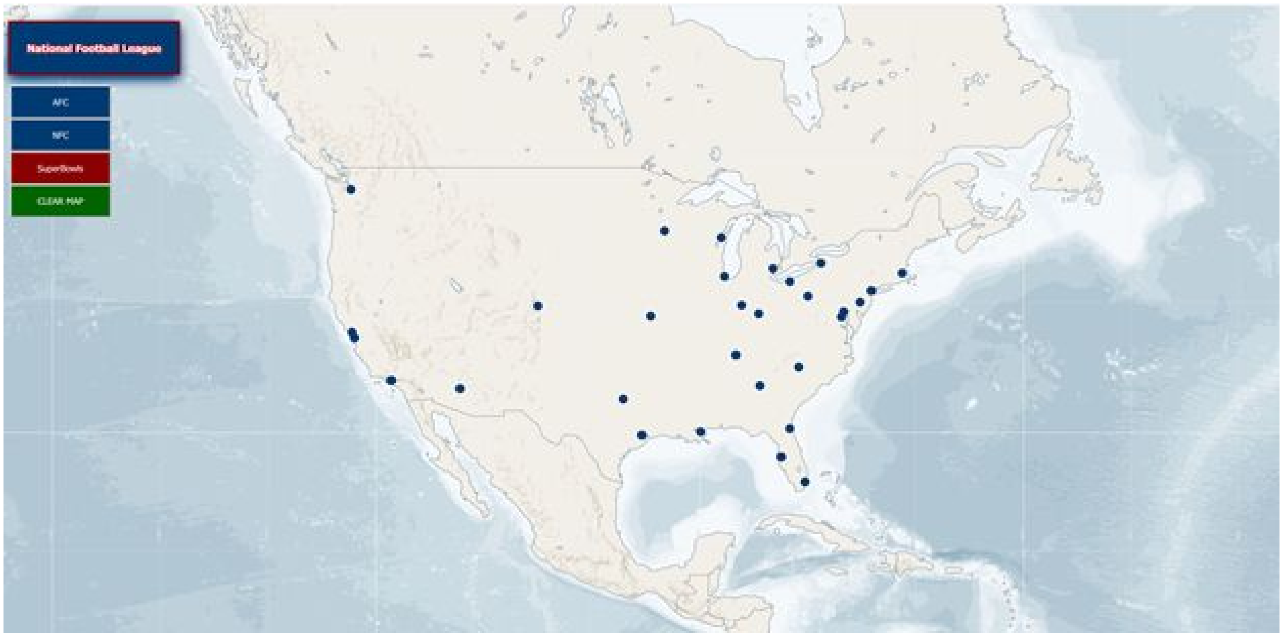
National Football League Map

By: J & R's Sports GIS



This map was created by J & R's sports analytics team members Jeff Goncer and Robert Bowman. Our goal is to provide our users with a dynamic, yet simple to use interactive map that displays information on our nation's favorite sports franchise, the NFL. The map gives the users the freedom to filter out layers so that they may display only the teams they wish to view. To spice up the map, we went further, adding information on all the Super bowls that have been played since championship number 1. If you read further in this documentation, we will explain the map in its entirety, including how the elements were designed.

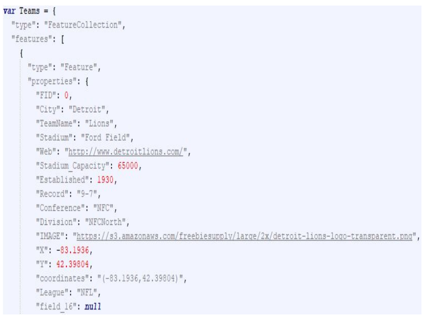
Mapbox/map design:

Mapbox is a software that is designed to help aide with problems that can be solved using mapping applications. The Javascript API gives developers the ability to manipulate Mapbox to serve the functions they need. Mapbox comes with many tutorials, however a knowledge of HTML, CSS and javascript will help the developer further understand how mapbox functions. Though mapbox is free to use initially, not everything in life truly comes free. After a map is viewed 50,000 times, the users account begins to be charged (something interesting to note if you are planning to use mapbox on a high traffic web page).

The map was designed using Mapbox styles. This gives developers the ability to manipulate the base map in whatever way they want. To set the basic cartogram color scheme, go to https://www.mapbox.com/cartogram. Once the cartogram is created , you can further edit the cartogram by using mapbox style Found at:https://www.mapbox.com/studio/styles/ . For our map we wanted to use a light background that wouldn't distract the user from the purpose of displaying our points. We directly manipulated a preset style given by the mapbox developers named North Star. This provided us with a clean map that doesn't distract users from what is important.

To eliminate errors and information generated by panning and zooming, we removed the ability to do so. This was done by calling three commands (map.scrollZoom.disable(), map.dragPan.disable(), map.doubleClickZoom.disable()).

Data Collection:

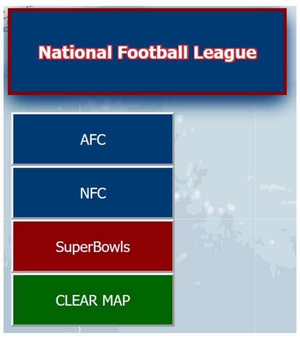


```
var Teams = {
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "FID": 0,
        "City": "Detroit",
        "TeamName": "Lions",
        "Stadium": "Ford Field",
        "Web": "http://www.detroitlions.com/",
        "Stadium_Capacity": 65000,
        "Established": 1930,
        "Record": "9-7",
        "Conference": "NFC",
        "Division": "NFCNorth",
        "IMAGE": "https://s3.amazonaws.com/freebiesupply/large/2x/detroit-lions-logo-transparent.png",
        "X": -83.1936,
        "Y": 42.39804,
        "coordinates": "(-83.1936,42.39804)",
        "League": "NFL",
        "field_16": null
```

Team information was gathered from selected websites. Information that was included were; 2017 W/L record, stadium, stadium capacity, when team was founded, city, team name, conference, division, and team website. The coordinates for the markers or points were acquired using ESRI ArcGIS using a city layer and creating points over this layer. After the team information and coordinates were recorded into an excel file it was then turned into a shapefile then into a geojson for map creation. QGIS was for the conversion of a shapefile into a json.

User interaction:

Buttons/css design:



The buttons were designed using CSS, while the onclick functions that set the layer filtering was done using HTML. The main button (Labeled National Football League) was color coded to represent the NFL's colors. When clicked this button removes any filters placed on the teams and sends a value to a Javascript function to display all Teams in the NFL. This all revolves around mapbox's built in function for filtering (.setfilter()). All buttons display their sub menu when the mouse pointer hovers over it. To set a filter value the button must be clicked.

```
<li><button class="buttonclass" onClick = "Javascriptfunction('X'','Y')">Text</button></li>
```

Above is an example of the button being called. The first step is to define with the button class that was created in the CSS. The next step is to define the onclick event. Set the onclick equal to the javascript function you want performed once the button is clicked(You may send the function Variables, EX: X,Y). The Text is what is displayed on the button as Text.

- Due to point overlap, we only allow superbowls to be displayed one at a time.

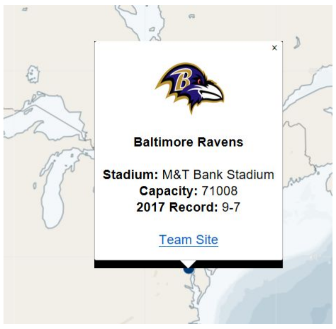
Button onClick Function:

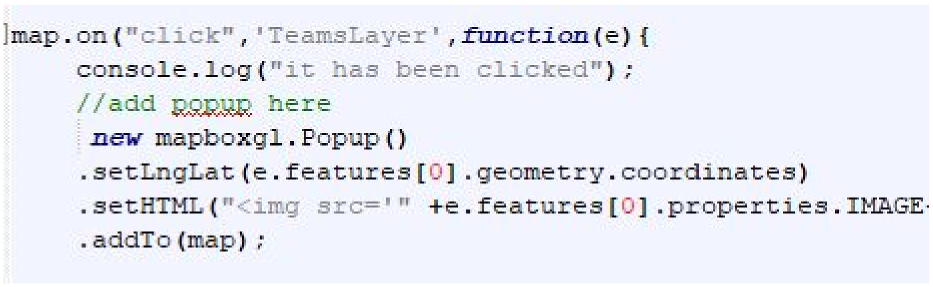
```
function myfunction(x,y)
        {
                Field = x;
                Search = y;
                map.setFilter("TeamsLayer", null);
                map.setFilter("TeamsLayer",["==",x,y]);
                map.setLayoutProperty('TeamsLayer', 'visibility', 'visible');
        };
```

This is how we created the filtering function. The function accepts two variables (x and y) and sets them as search variables. It is important to note that when searching in a Json, you need to have one variable for the field and one variable for the value you want to access. The first .setfilter clears any and all filters that may be present in the program. The second call creates a new filter and sets the filter = to the variables. The last line is how you can manipulate a mapbox layer's already defined properties. In this case we are turning the layer to visible to redisplay the points that were set to non-visible by another filter.

Pop ups:

An interaction function of our map are the popups for each marker. In each popup it displays the team logo, name, stadium, stadium capacity, 2017 W/L record, and a link that will take you to that individual teams website. Creating this pop up was taken from map box's how to create a heatmap tutorial. map.on("click",'TeamLayer',function(e){ creates a click function on the TeamLayer Json. ("click" can be replaced with mouse enter) This allows us to click on our markers that were created from our Json. console.log("it has been clicked") is optional and was used for testing clickability. new mapboxgl.Popup() is where the pop up creation starts, and the code under this is where it will be located and what will be inside the popup.





```
map.on("click",'TeamsLayer',function(e){
    console.log("it has been clicked");
    //add popup here
     new mapboxgl.Popup()
    .setLngLat(e.features[0].geometry.coordinates)
    .setHTML("<img src='" +e.features[0].properties.IMAGE
    .addTo(map);
```

.setLngLat(e.features[0].geometry.coordinates) takes the established coordinates created by the Json from the shapefile and uses this to place the popup box. There were two things that were included in the pop up that was approached differently, this includes having the team logo and the website be included in the popup. To add an image inside a pop up function it needs to be structured like this in the .setHTML line "<img src= ' " is telling the code that what we are calling is an image location (*link to the image must be included inside the Json*) +e.features[0].properties.IMAGE is calling the link and will show the logo in the popup. The information in the popup can be formatted by using breaks or "<br>" to separate the data. Adding a link inside a popup must be "<a href=' " +e.features[0].properties.Web +"'>Team Site</a>") href attribute specifies the URL of the page and creates a hyperlink. properties.Web is the location of the link inside the Json that it calls and Team Site is what will be shown in the popup.