Leaflet

Matthew Sisk

Updated November, 2023

Using Leaflet for dynamic maps

This is a brief overview of using the leaflet engine for dynamic maps

Leaflet is an open-source javascript library used for making dynamic web maps. It is used throughout the web as an easy way to create interactive maps using a variety of different data types. Here we will work with the R packages used for creating leaflet modules.

```
library(leaflet)
```

```
## Warning: package 'leaflet' was built under R version 4.3.2
```

```
library(sf)
```

```
## Warning: package 'sf' was built under R version 4.3.1
```

```
## Linking to GEOS 3.11.2, GDAL 3.6.2, PROJ 9.2.0; sf_use_s2() is TRUE
```

```
council.districts <-st_read("SampleData/City_Council_Districts.shp")
```

```
## Reading layer 'City_Council_Districts' from data source
##   'C:\Users\msisk1\Documents\GIT\teaching\Data-Viz-2023-Fall\Week03\Demonstrations\SampleData\City_C
##   using driver 'ESRI Shapefile'
## Simple feature collection with 6 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -86.36085 ymin: 41.59745 xmax: -86.19133 ymax: 41.76027
## Geodetic CRS:  WGS 84
```

```
facilities <- read.csv("SampleData/Public_Facilities.csv")
facilities.spatial <- facilities %>%
  st_as_sf(coords  = c("Lon","Lat"))%>%
  st_set_crs(value = 4326)
```

At its most basic, leaflet needs an initializing call to leaflet(), one or more basemaps to be added with addTiles() or addProvidorTiles() and then you can add anything else before finally printing the leaflet object to create it. Here we are using the pipe syntax so the initial leaflet() call is also serving to print it. Features can be added with from sf, sp, raster or data.frame objects.

```
leaflet()  %>%
  addTiles()  %>%
  addMarkers(data = facilities.spatial)
```

These are the most basic type of marker, but it is easy to customize these as well. The easiest thing is to create a popup with more information when you click on a marker. The format for this is html, so you can either just reference a field in the dataframe, or you can create a new string field with html and use that.

```
leaflet()  %>%
  addTiles()  %>%
  addMarkers(data = facilities.spatial, popup = ~POPL_NAME)
```

or you can create a new string field with html and use that.

```
facilities.spatial$popup <- paste("<b>",facilities.spatial$POPL_NAME,"</b><br>",
                                  "Type: ",facilities.spatial$POPL_TYPE,"<br>",
                                  "Phone: ",facilities.spatial$POPL_PHONE,sep ="")
leaflet()  %>%
  addTiles()  %>%
  addMarkers(data = facilities.spatial, popup = ~popup)
```

We can also change it so the markers are circles and control the color and size more directly. The base addMarkers() does not have a lot of customization options (although you can set icons on them). They also tend to overlap each other, so many maps make use of the alternate addCircleMarkers, which will let you customoze the size, opacity and color.

```
pal <- colorFactor(palette = 'Set1', domain =facilities.spatial$POPL_TYPE)

leaflet()  %>%
  addTiles()  %>%
  addCircleMarkers(data = facilities.spatial, popup = ~popup, color = ~pal(POPL_TYPE), stroke = 0, fill
```

Here, I used the colorFactor function to use a preset colorset, but you could also set them manually with something like palette = c("green", "yellow", "orange"). To see all of the presets in RColorBrewer, run RColorBrewer::display.brewer.all() in the console.

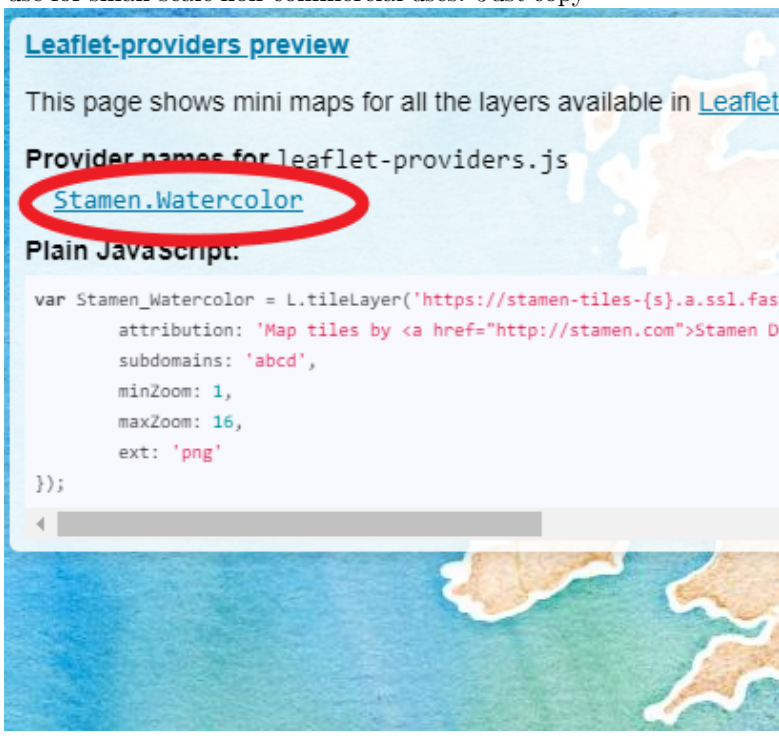
To find more about different colors you may want to use for maps (and to test for accessibility), you can check out the website that serves as the basis for the RColorBrewer package, ColorBrewer . This will let you test many of the palettes without having to load them into your maps and provides hex codes you can use directly in your palettes. Plus, ColorBrewer was developed by the researcher Cynthia Brewer, which is grade-a tool naming!

Basemap Tiles

We also have the option to use a wide variety of different basemaps beyond the default OpenStreetMap one by using addProviderTiles in place of addTiles. Then we can use the *providors* list that comes with the leaflet package to reference a wide variety of different tilesets. For example, the is the base imagery layer from ESRI.

```
leaflet()  %>%
  addProviderTiles(providers$Esri.WorldImagery)  %>%
  addCircleMarkers(data = facilities.spatial, popup = ~popup, color = ~pal(POPL_TYPE), stroke = 0, fill
```

You can find a list of all of the available provider tiles here: Leaflet Provider Demo. Some may require registration and an API key, but all of them should be free to use for small-scale non-commercial uses. Just copy the provider name and paste it after the providers$ image:



Grouping layers

We also have the ability in leaflet to interactively turn layers off and on by assigning them to groups and then using the Layer Control options. baseGroups will allow one to be selected, which overlayGroups will allow you to turn multiple layers off and on.

```
leaflet()  %>%
  addTiles(group = "Basic")  %>%
  addProviderTiles(providers$Esri.NatGeoWorldMap, group = "Nat Geo") %>%
  addCircleMarkers(data = facilities.spatial, popup = ~popup, color = ~pal(POPL_TYPE), stroke = 0, fill
  addLayersControl(
    baseGroups = c("Basic", "Nat Geo"),
    overlayGroups= c("Facilities"),
    options = layersControlOptions(collapsed = FALSE)
  )
```