Module4_Map_Reviewed

May 17, 2021

Maps

0.1 Table of Contents

Estimated Time Needed: 20 min

0.2 Overview

Among all the ways to visualize data, using maps is the best one if you have data that is related to spatial information, like the locations of the restaurants in a city. R has some packages that allow you to create the maps you always wanted, like maps (built-in), ggmaps and leaflet. Here, we will learn how to use the leaflet package.

[ ]:
```
install.packages("leaflet")
library(leaflet)
#Packages used to display the maps in this notebook
library(htmlwidgets)
library(IRdisplay)
```

Files used in this notebook Please run the code below to download the data sets into your workbench which we used in this notebook.

[ ]:
```
download.file("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
↪cloud/IBMDeveloperSkillsNetwork-DV0103EN-SkillsNetwork/labs/module%201/
↪countries.txt",
              destfile = "/resources/data/countries.txt", quiet = TRUE)
download.file("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
↪cloud/IBMDeveloperSkillsNetwork-DV0103EN-SkillsNetwork/labs/module%201/
↪energy.csv",
                    destfile = "/resources/data/energy.csv", quiet = TRUE)
```

##

Creating a map

Basically, it takes three steps to make a map visualization: 1. Create a map widget 1. Customize your map settings and add content (layers) to the map widget using layer functions 1. Display and enjoy your map

First, let's see how to create the map widget.

0.2.1 Creating a map widget

The creation of a map widget is pretty simple and straight forward. You just need to call the function leaflet().

[ ]:
```
map <- leaflet()
```

After the function call, you will have an object that represents an empty world map: it will be an empty space that follows the same coordinate system used in all world maps.

One interesting point about the leaflet() function is that it has an optional parameter called data. This parameter exists so you can pass spatial data to your map (like the coordinates of your town) and use it when adding the layers. If you pass a data frame as your source of spatial data, there are two ways of accessing and using that information in the layer functions: the first one is done automatically if your data frame has columns named as *lat/latitude* and *lng/long/longitude* (case insensitive); the second way is to explicitly declare the latitude and longitude values when callng the layer functions.

Here is an example of a map widget creation with the built-in quakes data frame from R.

[ ]:
```
map <- leaflet(quakes)
```

0.2.2 Customizing the map

Before talking about how to customize your maps, there is an interesting tool that you should discover first: the pipe operator %>% from magrittr package. Basically, it passes whatever is on its left side as the first parameter of the function that comes on its right side and then returns the result. When working with the leaflet package, the pipe operator is very useful, since most of the functions in this package need a map as its first parameter and return a new map. However, since the operator just returns the result of the function on its right, your new map will be displayed but will not be recorded unless you use an assignment statement, just like you do for other functions. While studying the first customization functions, we will have examples both with and without the pipe operator but, later on, we will just use the pipe operator.

##

Basic settings

After creating your map, the first thing you can do is set the first view of your map and it can be done in two ways. ##### Setting the first view The first way is by calling the function setView(map, lng, lat, zoom), which will define the center of the first view and the zoom.

```
[ ]: #Using the pipe operator
     map <- leaflet() %>% setView(lng = 86.92, lat = 27.99, zoom = 5)
     #Without the pipe operator
     map <- leaflet()
     map <- setView(map, lng = 86.92, lat = 27.99, zoom = 5)
```

Setting the first view using bounds The second way is by setting the bounds of the first view by calling the function fitBounds(map, lng1, lat1, lng2, lat2). This will fit your first view inside the rectangle defined by the coordinates you gave as arguments.

```
[ ]: #Using the pipe operator
     map <- leaflet() %>% fitBounds(86.8, 27.9, 87, 28)
     #Without the pipe operator
     map <- leaflet()
     map <- fitBounds(map, 86.8, 27.9, 87, 28)
```

##

Adding base map

Even after setting the first view, the map will still be empty, since we have not added a base yet. The most common way to do it is by calling the addTiles(map) function, which uses the OpenStreetMap by default.

```
[ ]: #Using default base map
     map1 <- leaflet() %>% fitBounds(86.8, 27.9, 87, 28) %>% addTiles()
     saveWidget(map1, file="map1.html", selfcontained = F)
     display_html(paste("<iframe src=' ", 'map1.html', " ' width='100%'␣
      ↪height='300'","/>"))
```

You can also use third-party tiles by calling addProviderTiles(map, tile) (you can find some tiles here http://leaflet-extras.github.io/leaflet-providers/preview/index.html). It is also possible to combine multiple tiles by exploring the opacity parameter of those functions, so feel free to play with it.

```
[ ]: #Using a third-party base map
     map2 <- leaflet() %>% fitBounds(86.8, 27.9, 87, 28) %>%␣
      ↪addProviderTiles("Stamen.Watercolor")
     saveWidget(map2, file="map2.html", selfcontained = F)
     display_html(paste("<iframe src=' ", 'map2.html', " ' width='100%'␣
      ↪height='300'","/>"))
```

##

Adding markers

It is possible to add markers on your map to highlight a special location. To do so, you just need to call the function addMarkers(map, lng, lat). If during the map creation you gave the spatial data, ALL that data will be automatically read and marked on the map if you don't specify the parameters *lng* and *lat*.

```
[ ]: map3 <- leaflet(quakes) %>% addTiles(
                      ) %>% addMarkers(lng = quakes$long[1:10],
                                       lat = quakes$lat[1:10])
     saveWidget(map3, file="map3.html", selfcontained = F)
     display_html(paste("<iframe src=' ", 'map3.html', " ' width='100%'␣
      ↪height='300'","/>"))
```

Changing the markers It is also possible to change the marker. You can call addCircleMarkers() to have circular markers or create your own markers using the function makeIcon(image_file, height, width) and passing it as the *icon* argument in the addMarkers() function.

```
[ ]: #Circle markers examples
     map4 <- leaflet(quakes) %>% addTiles(
                      ) %>% addCircleMarkers(lng = quakes$long[1:10],
                                             lat = quakes$lat[1:10])
     saveWidget(map4, file="map4.html", selfcontained = F)
     display_html(paste("<iframe src=' ", 'map4.html', " ' width='100%'␣
      ↪height='300'","/>"))
```

Popups You can also enable popups for your markers so, when they are clicked, some content is displayed by passing this content as the popup argument in the addMarkers() call.

```
[ ]: map5 <- leaflet() %>% fitBounds(86.8, 27.9, 87, 28) %>% addTiles(
                ) %>% addMarkers(lng = 86.92, lat = 27.99, popup = "Mount Everest")
     saveWidget(map5, file="map5.html", selfcontained = F)
     display_html(paste("<iframe src=' ", 'map5.html', " ' width='100%'␣
      ↪height='300'","/>"))
```

Clustering markers If we try to use all the quakes data, the map will become unreadble due to the huge number of markers in the same region. When this happens, you can call the clustreOptions parameter from add markers, which will group your markers by region and display the number of markers in each region.

```
[ ]: #Without clustering the markers
     map6 <- leaflet(quakes) %>% addTiles() %>% addCircleMarkers()
     saveWidget(map6, file="map6.html", selfcontained = F)
     display_html(paste("<iframe src=' ", 'map6.html', " ' width='100%'␣
      ↪height='300'","/>"))
     #Clustering the markers
     map7 <- leaflet(quakes) %>% addTiles() %>% addMarkers(clusterOptions =␣
      ↪markerClusterOptions())
     saveWidget(map7, file="map7.html", selfcontained = F)
```

```
display_html(paste("<iframe src=' ", 'map7.html', " ' width='100%'␣
↪height='300'","/>"))
```

##

Adding shapes

Another possible way to customize your map is by adding shapes like circles, rectangles or any kind of polygon you can imagine. First, let's talk about adding circles. ##### Circles Circles can be added by calling the function addCircles() and they are very similar to the circle markers, with the only difference being that the circle markers have its radius given in pixels while the circles have its radius in meters, which mean that the circles are rescaled with the whole map while the markers have a constant size.

```
[ ]: #Try changing the zoom to see the difference between circles and circle markers
map8 <- leaflet(quakes) %>% addTiles() %>% addCircleMarkers(lng = quakes$long[1:
↪5],
                lat = quakes$lat[1:5]) %>% addCircles(lng = quakes$long[5:10],
                                                        lat = quakes$lat[5:10],␣
↪color = 'red')
saveWidget(map8, file="map8.html", selfcontained = F)
display_html(paste("<iframe src=' ", 'map8.html', " ' width='100%'␣
↪height='300'","/>"))
```

Rectangles If you want to highlight an area, adding a rectangle may be what you need. To do so, just call the function addRectangles() and pass the coordinates of two points, which will be the delimiters of your rectangle.

```
[ ]: map9 <- leaflet() %>% addTiles() %>% addMarkers(lng = 86.92, lat = 27.99,
                                  popup = "Mount Everest") %>% addRectangles(86.
↪9, 27.95, 87, 28.05)
saveWidget(map9, file="map9.html", selfcontained = F)
display_html(paste("<iframe src=' ", 'map9.html', " ' width='100%'␣
↪height='300'","/>"))
```

Polygons with matrix Now, if you want to add a polygon, you will have to call addPolygons() and give the polygon information. One way to give this information is by creating a matrix with rows (lng, lat) where each row correspond to a vertice of your polygon. If want more than one polygon, separate them in the matrix with a NA row.

```
[ ]: tri <- matrix(c(86.87, 27.95, 86.97, 27.95, 86.92, 28.05), ncol = 2, byrow =␣
↪TRUE)
map10 <- leaflet() %>% addTiles() %>% addMarkers(lng = 86.92, lat = 27.99,
                            popup = "Mount Everest") %>% addPolygons(lng = tri[, 1],
                                                            lat = tri[, 2])
saveWidget(map10, file="map10.html", selfcontained = F)
display_html(paste("<iframe src=' ", 'map10.html', " ' width='100%'␣
↪height='300'","/>"))
```

Polygons with sp object Another way to give the polygon information is by passing a sp object (from the sp package) with that data. A great example of this is to plot the countries borders in the map.

[ ]:
```
#This example uses the library rgdal to read a geojson file
#with the countries' borders information and convert it to
#a sp object
library(rgdal)
countries <- readOGR("/resources/data/countries.txt", "OGRGeoJSON")
map11 <- leaflet(countries) %>% addTiles() %>% addPolygons(weight = 1)
saveWidget(map11, file="map11.html", selfcontained = F)
display_html(paste("<iframe src=' ", 'map11.html', " ' width='100%'␣
 ↪height='300'","/>"))
```

##

Colors and legends

When creating your own map, you can show your data through colors and, to do so, you will have to create a color object according to your data. The most useful color objects you can create are:

- colorNumeric
- colorFactor

They are, basically, created the same way, in the sense that both of them require a palette (the color scale you want to use; check http://colorbrewer2.org/ for some palette options) and a domain (the values to match with the colors). After adding the color, you can add legends to improve your map visualization. To do so, you call the function addLegend(map, position, palette, values) where palette and values are the same you used to create the color object.

colorNumeric The main use for colorNumeric is to represent continuous data, since it match the domain interval to the palette interval.

[ ]:
```
energy <- read.csv("/resources/data/energy.csv")
#merge the energy data frame with countries
countries <- sp::merge(countries, energy[, c(1, 3)], by = "geounit", all.x = T)
color <- colorNumeric("YlOrRd", energy$Value)
map12 <- leaflet(countries) %>% addTiles() %>% addPolygons(stroke = FALSE,
                            fillColor = ~color(Value),
                            fillOpacity = 1) %>% addLegend("topright",
                                                   pal = color,
                                                   values =␣
 ↪countries@data$Value,
                                                   title = "kWh per capita",
                                                   opacity = 1)

saveWidget(map12, file="map12.html", selfcontained = F)
display_html(paste("<iframe src=' ", 'map12.html', " ' width='100%'␣
 ↪height='300'","/>"))
```

colorFactor Just like it sounds, colorFactor is better used to represent factors or categorical data.

```
[ ]: color <- colorFactor("Set1", countries$continent)
     map13 <- leaflet(countries) %>% addTiles() %>% addPolygons(stroke = F,
                                  fillColor = ~color(continent),
                                  fillOpacity = 1) %>% addLegend("bottomleft",
                                                                pal = color,
                                                                values = ~continent,
                                                                opacity = 1,
                                                                title = "Continent")
     saveWidget(map13, file="map13.html", selfcontained = F)
     display_html(paste("<iframe src=' ", 'map13.html', " ' width='100%'␣
      ↪height='300'","/>"))
```

0.2.3 About the Author:

Hi! It's Martha Aghili the author of this notebook. I hope you found R easy to learn! There's lots more to learn about R but you're well on your way. Feel free to connect with me if you have any questions.