

Displaying Interactive Map with Shiny App

```
## rgdal: version: 1.2-13, (SVN revision 686)
##   Geospatial Data Abstraction Library extensions to R successfully loaded
##   Loaded GDAL runtime: GDAL 2.2.0, released 2017/04/28
##   Path to GDAL shared files: C:/Users/Ruiqi/Documents/R/win-library/3.4/rgdal/gdal
##   Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##   Path to PROJ.4 shared files: C:/Users/Ruiqi/Documents/R/win-library/3.4/rgdal/proj
##   Linking to sp version: 1.2-5
```

```
##
## Attaching package: 'dplyr'
```

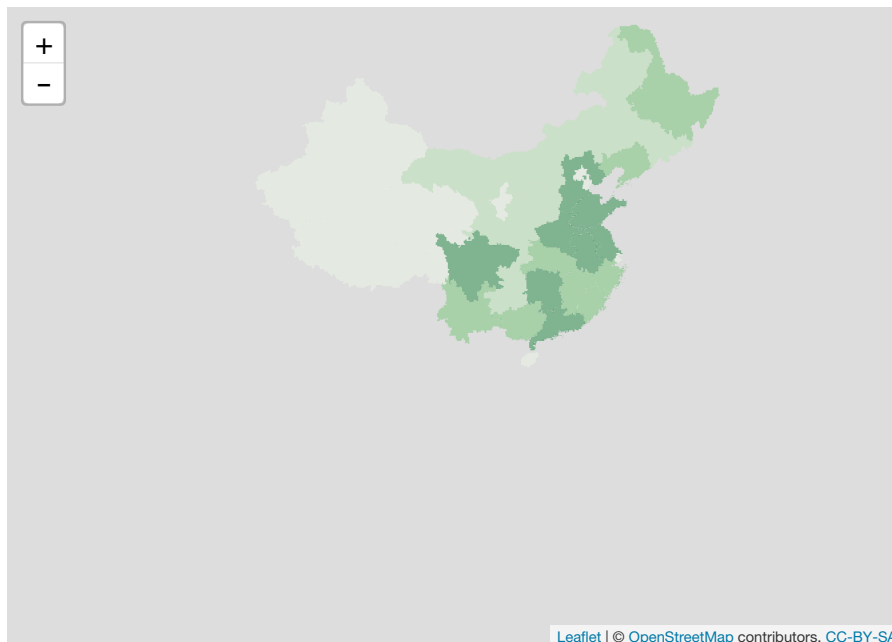
```
## The following objects are masked from 'package:stats':
##
##   filter, lag
```

```
## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:/Users/Ruiqi/stat133/stat133-hws-fall17/POST2", layer: "CHN_adm1"
## with 31 features
## It has 12 fields
## Integer64 fields read as strings:  ID_0 ID_1 CCN_1
```

I. INTRODUCTION

In Post 1 I wrote about how to construct a map with data on `r` using package “`leaflet`”. Soon after, I realize that one of the essential advantage of `leaflet` map like the one below is that the package let people interact with the map, where they can zoom in and out to any places on the map. But only having an output in the form of PDF, picture, or even html can't really utilize its interactive nature. Then, I thought of using the interactive agent, shiny app, as the output form of a `leaflet` map; so this becomes today's POST's topic: Displaying Interactive Map with Shiny App.



In the following paragraphs, I will first talk about how to transplant `leaflet` map into shiny map, and furthermore, how we can make advanced use of the interactive nature of shiny to better display and analyze the data.

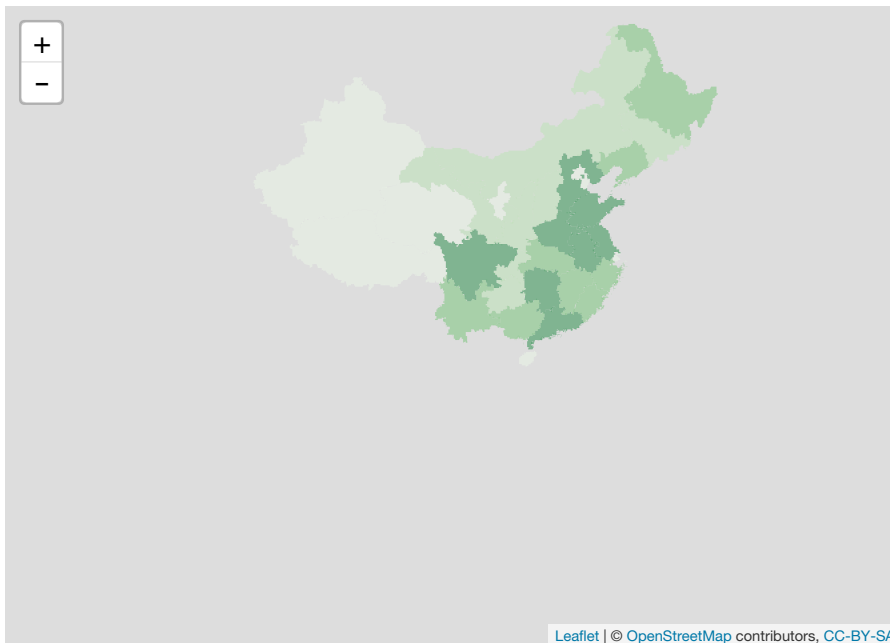
The data we are using today is the population of 31 provinces in China in 2015. Also the code of making the map are displayed below.

```
library(shiny)
library(leaflet)
library(sp)
library(rgdal)
library(RColorBrewer)

my_spdf=readOGR( dsn= "C:/Users/Ruiqi/stat133/stat133-hws-fall17/POST2" , layer="CHN_adm1")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:/Users/Ruiqi/stat133/stat133-hws-fall17/POST2", layer: "CHN_adm1"
## with 31 features
## It has 12 fields
## Integer64 fields read as strings:  ID_0 ID_1 CCN_1
```

```
CHN <- read.csv("C:/Users/Ruiqi/stat133/stat133-hws-fall17/POST2/CHNLocations2.csv",header = TRUE)
DF2 <- merge.data.frame(CHN,my_spdf@data)
my_spdf@data <- DF2
m=leaflet(my_spdf)%>% addTiles() %>% setView( lat=23, lng=100 , zoom=3) %>%
  addPolygons( stroke = FALSE, fillOpacity = 0.5, smoothFactor = 0.5, color = ~colorQuantile("Greens", pop2015)(po
p2015) )
m
```



II. Basic: Putting the Map on Shiny Our first task is to move the code above into the shiny app.

To do so, let's first look at the Ui and Server individually.

###UI: We first call out leaflet map using leafletOutput(), in the brackets we first need to give our map a name. hmmm let's make it short and simple, so it's easier for us to call on it later.

"Ruiqizhaopost2fullmarksmap" sounds pretty just right.

```
ui <- fluidPage(
  leafletOutput(outputId = "Ruiqizhaopost2fullmarksmap") )
```

Then, we need to adjust the size of the map to fit with the web window. Personally, I prefer to let the map stretch out to the whole screen. So simple as below. All we need to do is add the width and height of your choice.

```
ui <- fluidPage(
  leafletOutput("Ruiqizhaopost2fullmarksmap",width = 1240,height = 1000))
```

SERVER:

Having decided the layout of the map, we then move onto the server side.

All we have to do here is wrap the map function with the function renderLeaflet({}) and assign it back to the short and simple ID we gave our map in the previous part.

```
server <- function(input, output) {
  output$Ruiqizhaopost2fullmarksmap <- renderLeaflet({
    leaflet(my_spdf)%>% addTiles() %>% setView( lat=30, lng=121 , zoom=4) %>%
      addPolygons( stroke = FALSE, fillOpacity = 0.5, smoothFactor = 0.5, color = ~colorQuantile("Greens", pop2015
)(pop2015) )
  })
}
```

After filling the server side, the only step left before creating out interactive map is integrate the Ui and Server.

And you can use the browser or small pop up window to see your shiny app with the map.

```
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Our application of leaflet with shiny are not only this shown above. More importantly, Shiny's interactive feature enables everyone to change how the data and the map can be displayed. From now on, we are gonna look at several variables we can add to our shiny app.

III. Advanced: Further develop the interactiveness

Changing Map Tiles

The map package leaflet offers many map tiles. For example, here we have two tiles called "Esri.WorldImagery"(the Satellite map) and the "Openstreetmap". For the two cases we can use addPolygons and addProviderTiles to add then separately. And because we are operating under shiny now, we can add the control button for switching the layers using "addLayersControl()" as below.

```
leaflet(my_spdf)%>% addTiles() %>% setView( lat=30, lng=121 , zoom=4) %>%  
  addPolygons( stroke = FALSE, fillOpacity = 0.5, smoothFactor = 0.5, color = ~colorQuantile("Greens", pop2015  
(pop2015) ) %>%  
  
  addTiles(options = providerTileOptions(noWrap = TRUE), group="Openstreetmap") %>%  
  addProviderTiles("Esri.WorldImagery", group="Satellitemap") %>%  
  addLayersControl(baseGroups = c("Openstreetmap", "Satellitemap"), position="topleft", options = layersControlOptions(collapsed = FALSE))
```



The result map and its function are displayed below

```
ui <- fluidPage(  
  leafletOutput("Ruiqizhaopost2fullmarksmmap", width = 1240, height = 1000)  
)  
  
server <- function(input, output) {  
  leaflet(my_spdf)%>% addTiles() %>% setView( lat=30, lng=121 , zoom=4) %>%  
    addPolygons( stroke = FALSE, fillOpacity = 0.5, smoothFactor = 0.5, color = ~colorQuantile("Greens", pop2015  
(pop2015) ) %>%  
    addTiles(options = providerTileOptions(noWrap = TRUE), group="Openstreetmap") %>%  
    addProviderTiles("Esri.WorldImagery", group="Satellitemap") %>%  
    addLayersControl(baseGroups = c("Openstreetmap", "Satellitemap"), position="topleft", options = layersControlOptions(collapsed = FALSE))  
}  
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Changing the map tile is easier than adding other variables, because `addlayerscontrol` is nested in the `leaflet` map function and thus we only need to make changes to the server not UI. Besides, all the below adjustment to our interactive map include altering both UI and Server.

Changing Base Color

Notice that in previous cases, the color of data is different degree of green, representing the size of data. So, can we set up a option and let whomever view the shiny change the color degree they want? To achieve this we first look back at our ui.

I personally would like to put a selection menu at the top right. So first we create a `absolutePanel` that can contain many inputs. then we include a title and the function `selectInput`, in which we assign the input a ID and fill out the choices with `rownames` of colors.

```
ui <- fluidPage(  
  leafletOutput("Ruiqizhaopost2fullmarksmap",width = 1240,height = 1000)  
,  
  absolutePanel(top = 10,right=10,  
    titlePanel("Population by Provinces"),  
    selectInput("colors", "Color Scheme",  
      rownames(subset(brewer.pal.info, category %in% c("seq", "div")))  
    )))
```

That's how we create the select menu in UI. We make every choice in the select menu equal the `rownames` of `brewer.pal` info. And then we move on to Server to put the `input$color` at where I originally put "Greens"

```
server <- function(input, output) {  
  leaflet(my_spdf)%>% addTiles() %>% setView( lat=30, lng=121 , zoom=4) %>%  
    addPolygons( stroke = FALSE, fillOpacity = 0.5, smoothFactor = 0.5, color = ~colorQuantile(input$colors, pop  
2015)(pop2015) )%>%  
    addTiles(options = providerTileOptions(noWrap = TRUE), group="Openstreetmap")%>%  
    addProviderTiles("Esri.WorldImagery", group="Satellitemap") %>%  
    addLayersControl(baseGroups = c("Openstreetmap","Satellitemap"), position="topleft",options = layersControlOptions(collapsed = FALSE))  
}  
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

Showing data of selected Province

It's not so convenient to throw the whole table of population of each provinces right into the face of the viewers. So it would be better if we could display the data they really want to see. In this way, we can make a selection bar attached with a small table showing the data of the selected provinces. On the UI aspect, we need to add a select input and a table output like the one below.

```

ui <- fluidPage(
  leafletOutput("map1",width = 1240,height = 1000),
  absolutePanel(top = 10,right=10,
    titlePanel("Population by Provinces"),
    selectInput("colors", "Color Scheme",
      rownames(subset(brewer.pal.info, category %in% c("seq", "div"))))
  ),
  selectInput("Provs", "Provinces", choices = my_spdf@data$NAME_1),
  tableOutput("table1")
)

```

And on the server side, we only need to add the following code to link the input with the output.

```

server <- function(input, output) {
  output$table1 <- renderTable({fn[fn$NAME_1==input$Provs,]})
}

```

Showing provinces with population above xxx size

Now what if we want to offer the view a choice of simply viewing only the provinces with population above certain level? We can also achieve it using Shiny conveniently. For the server, we would like to add a slider input.

```

ui <- fluidPage(
  leafletOutput("map1",width = 1240,height = 1000),
  absolutePanel(top = 10,right=10,
    titlePanel("Population by Provinces"),
    sliderInput("pop", "Provinces with population size greater than",
      min = 0,
      max = 120,
      value = 0),
    selectInput("colors", "Color Scheme",
      rownames(subset(brewer.pal.info, category %in% c("seq", "div"))))
  ),
  selectInput("Provs", "Provinces", choices = my_spdf@data$NAME_1),
  tableOutput("table1")
)

```

And for the server, it's a bit more complicated than previous ones. Personally, I first assigned those provinces with lower than selected population to be NA, so that they will appear on the map as gray areas using bracket selection. And then do the same treatment as before to the data left.

```

server <- function(input, output) {
  output$table1 <- renderTable({fn[fn$NAME_1==input$Provs,]})
  output$map1 <- renderLeaflet({
    my_spdf@data$pop2015[my_spdf@data$pop2015<input$pop] <- NA
    m=leaflet(my_spdf)%>% addTiles() %>% setView( lat=30, lng=121 , zoom=4) %>%
      addPolygons( stroke = FALSE, fillOpacity = 0.5, smoothFactor = 0.5, color = ~colorQuantile(input$colors, pop
2015)(pop2015) )%>%
      addMarkers(lng = 116,
        lat = 39.963,
        popup = "You are not here.",
        options = markerOptions(draggable = TRUE, riseOnHover = TRUE))%>%
      addTiles(options = providerTileOptions(noWrap = TRUE), group="Openstreetmap")%>%
      addProviderTiles("Esri.WorldImagery", group="Satellitemap") %>%
      addLayersControl(baseGroups = c("Openstreetmap", "Satellitemap"), position="topleft", options = layersControlOptions(collapsed = FALSE))
    m
  })
}

shinyApp(ui = ui, server = server)

```

Shiny applications not supported in static R Markdown documents

So afterall, the shiny app code would look like this below.

```
library(shiny)
library(leaflet)
library(sp)
library(rgdal)
library(RColorBrewer)
library(dplyr)

my_spdf=readOGR( dsn= "C:/Users/Ruiqi/stat133/stat133-hws-fall17/POST2" , layer="CHN_adm1")

## OGR data source with driver: ESRI Shapefile
## Source: "C:/Users/Ruiqi/stat133/stat133-hws-fall17/POST2", layer: "CHN_adm1"
## with 31 features
## It has 12 fields
## Integer64 fields read as strings: ID_0 ID_1 CCN_1

CHN <- read.csv("C:/Users/Ruiqi/stat133/stat133-hws-fall17/POST2/CHNLocations2.csv",header = TRUE)
DF2 <- merge.data.frame(CHN,my_spdf@data)
my_spdf@data <- DF2
name <- select(my_spdf@data,1)
popsize <- select(my_spdf@data,4)
la <- select(my_spdf@data,2)
ln <- select(my_spdf@data,3)
fn <- cbind(name,popsize,la,ln)

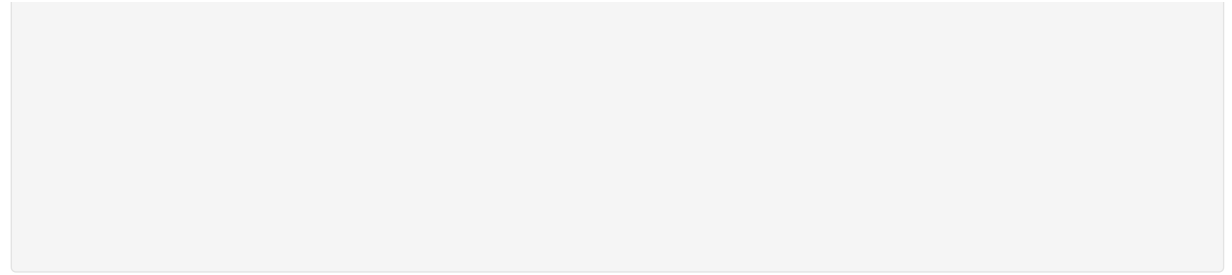
ui <- fluidPage(
  leafletOutput("map1",width = 1240,height = 1000),
  absolutePanel(top = 10,right=10,
    titlePanel("Population by Provinces"),
    sliderInput("pop", "Provinces with population size greater than",
      min = 0,
      max = 120,
      value = 0),
    selectInput("colors", "Color Scheme",
      rownames(subset(brewer.pal.info, category %in% c("seq", "div"))
    ),
    selectInput("Provs", "Provinces",choices = my_spdf@data$NAME_1),
    tableOutput("table1")

  )
)

server <- function(input, output) {
  output$table1 <- renderTable({fn[fn$NAME_1==input$Provs,]})
  output$map1 <- renderLeaflet({
    my_spdf@data$pop2015[my_spdf@data$pop2015<input$pop] <- NA
    m=leaflet(my_spdf)%>% addTiles() %>% setView( lat=30, lng=121 , zoom=4) %>%
      addPolygons( stroke = FALSE, fillOpacity = 0.5, smoothFactor = 0.5, color = ~colorQuantile(input$colors, pop
2015)(pop2015) )%>%
      addMarkers(lng = 116,
        lat = 39.963,
        popup = "You are not here.",
        options = markerOptions(draggable = TRUE, riseOnHover = TRUE))%>%
      addTiles(options = providerTileOptions(noWrap = TRUE), group="Openstreetmap")%>%
      addProviderTiles("Esri.WorldImagery", group="Satellitemap") %>%
      addLayersControl(baseGroups = c("Openstreetmap", "Satellitemap"), position="topleft",options = layersControlOptions(collapsed = FALSE))
    m
  })
}

shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents



IV. **Conclusion** After this POST I hope everyone could learn how to display a interactive map using shiny, because as the globalization going on, the probability we would make use geological data would for sure increase. And thus I believe this would be extremly beneficial to most of us in the future. Thanks.