Leaflet

- Leaflet is one of the most popular open-source JavaScript libraries for interactive maps. It's used by websites you are likely to encounter regularly.
- It allows for interactive panning/zooming
- We can use this in conjunction with shiny to create an even more interesting user experience.
- The Basic steps:
  1. Create a map widget by calling $leaflet()$.
  2. Add layers (i.e., features) to the map by using layer functions (e.g. addTiles, addMarkers, addPolygons) to modify the map widget. The pipe operator % > % for leaflet can be thought of as the + sign in ggplot. That is, it will add upon existing layers of your map.
  3. Print the map widget to display it.

Example 1

– Let us map our current location in Allbritton (222 Church Street)

```
require(leaflet)
leaflet()  %>%
  addTiles() %>%
  addMarkers(lng=-72.655974, lat=41.554203,
popup="Data Viz Class")
```

– Next, suppose I would like to visually display where I spend most of my time:

```
my_time<-data.frame(location=c("Usdan", "Allbritton", "North College"),
                    latitude=c(41.556770,41.554203, 41.556320),
                    longitude=c(-72.656868,-72.655974, -72.656157),
                    time=c(0.15,0.8,0.05))

leaflet(my_time)  %>%
  addTiles() %>%
addMarkers(lng=-72.655974, lat=41.554203,
popup="Data
Viz Class") %>%
  addCircles(lng=~longitude,
             lat=~latitude,
             radius=my_time$time*100,
             popup=my_time$location)

## The problem with the above code is that
## the area of the circle is how the
## eye will compare differences, not by the radius
## Adapt this by solving A=pi*r^2 for r.

leaflet(my_time)  %>%
  addTiles() %>%
addMarkers(lng=-72.655974, lat=41.554203, popup="Data
Viz Class") %>%
  addCircles(lng=~longitude,
             lat=~latitude,
```

```
              radius=sqrt(my_time$time/pi)*100,
              popup=my_time$location)
```

  - Now you think about 6 or so locations you frequent on campus. How would you make a similar plot of your average Monday?

- Adding data. The package provides 3 options for using real, large datasets in leaflet:
  - Data from R: data frames (our typical files)
  - Data from sp: (such as spatialPoints and SpatialPolygons)
  - Data from maps
- Suppose I want to map all of the Dunkin' Donuts in Connecticut

```
ctlist <- read.csv("~/Desktop/ctlist.csv")

leaflet(ctlist) %>%
  addTiles() %>%
  #addProviderTiles(providers$Thunderforest.SpinalMap) %>%
  setView(-72.690940, 41.651426, zoom = 8) %>%
  addCircles(~lng, ~lat,
             popup=ctlist$address,
             weight = 3, radius=40,
             color="orange", stroke = TRUE,
             fillOpacity = 0.8) %>%
  addLegend("bottomright", colors="orange",
            labels="Dunkin", title="In Connecticut")
```

- You can change the tiles (basemap). For a full list you can visit: https ://rstudio.github.io/leaflet/basemaps.html
- Recall our college enrollment data for states in the midwest.

```
college_enrollment <- read.csv("~/Desktop/college_enrollment.csv")

leaflet(college_enrollment) %>%
  addTiles() %>%
  addCircles(~long, ~lat,
             popup=college_enrollment$name,
             weight = 3,
             radius=sqrt(college_enrollment$enrollment/pi)*300,
             color="black", stroke = FALSE,
             fillOpacity = 0.8)
```

Shiny Integration

- Leaflet integrates naturally into Shiny

```
require(shiny)

ui<-fluidPage(
  sliderInput(inputId="bubblesize",
              label="What multiplier is needed?",
              min=0,
```

```
              max=1000,
              value=50,
              step=50),
  leafletOutput(outputId="myMap")
)

server<-function(input, output){
  college_enrollment <- read.csv("~/Desktop/college_enrollment.csv")
  output$myMap<-renderLeaflet(

leaflet(college_enrollment) %>%
  addTiles() %>%
  addCircles(~long, ~lat,
             popup=college_enrollment$name,
             weight = 3,
             radius=sqrt(college_enrollment$enrollment/pi)*input$bubblesize,
             color="black", stroke = FALSE,
             fillOpacity = 0.8)
  )

}

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

Choropleths in Leaflet

- Great news! Choropleths in Leaflet are quite nice!
- There are some different obstacles you will face here. You need to extract the data from the Spatial Polygon Data Frame, add attribute information, and then place it back in the Spatial Polygon Data Frame in the same order you took it out.

```
require(rgdal)
require(leaflet)
require(geojsonio)
require(readr)
require(data.table)
require(dplyr)
require(RColorBrewer)
require(shiny)

#Import data!
Obese <- read_csv("P:/QAC/qac251/course_materials/Code and Data/Obese.csv")

#Import geojson file
#geojson_read will read in a ".json" file
#and create a Large Spatial Polygons DataFrame

state_shapes<-geojson_read("https://raw.githubusercontent.com/PublicaMundi/MappingAPI/master/data/g

#Explore Spatial Polygons DataFrame
slotNames(state_shapes)

# Save the data row.names into an explicit variable
state_shapes$rn <- row.names(state_shapes)

# Create temporary data tables to work on the attributes
tmp.states <- data.table(state_shapes@data)
tmp.states<-mutate(tmp.states, name=tolower(name))
Obese<-mutate(Obese, name=tolower(State))
out.states<-merge(tmp.states, Obese, by="name", all.x=TRUE) #all.x=TRUE Keeps all information from
out.states<-data.table(out.states)

# Then let's re-attach the table to the original SpatialPolygons DataFrame
#(preserving the original order of the row.names)
setkey(out.states, rn)
state_shapes@data <- out.states[row.names(state_shapes)]


#Decide on a color palette and how many
#color levels you want.
#Keep domain=NULL, unless you have an
#exact specification of how you want the colors
#to be divided



pal <- colorQuantile("RdPu",domain=NULL, n =5)
```

```
leaflet(data = state_shapes) %>%
      addProviderTiles("CartoDB.Positron") %>%
      addPolygons(fillColor = ~pal(Children),
                  fillOpacity = 0.8,
                  color = "black",
                  weight = 1,
                  popup=~paste(name, "<br>Rate:", Children) ) %>%
      addLegend("bottomleft",
                colors=brewer.pal(5,"RdPu"),
                labels=c("low","","","","high"),
                title="Relative Obesity Rates")
```

- Now let us work on a chorpleth at the country-level

```
#Load in country-level geojson file
country_shapes<-geojson_read("https://raw.githubusercontent.com/johan/world.geo.json/master/countri

#Import attribute-level data
sulferdioxide <- read_csv("P:/QAC/qac251/course_materials/Code and Data/sulferdioxide.csv")


#The row names contain valuable ID information. I do not want
#to lose this information so we will create a new variable
# called `rn` (I use this to denote rownames)
country_shapes$rn <- row.names(country_shapes)

#I create a temporary data.table to manipulate.
#It is recommended that you do not try to manipulate the data
#directly in a Large spatial polygon data.frame.
#Polygon ID information and labels get lost easily
country_temp<-data.table(country_shapes@data)

#As before, we need to make sure that the attribute data
#can be matched to the polygon data.
#The only difference I notice is the capitalization and id names.

#Let us have ID in both data sets equal to `name`
country_temp<-mutate(country_temp, name=tolower(name))
names(sulferdioxide)[2]<-"name"
sulferdioxide<-mutate(sulferdioxide, name=tolower(name))

#Merge your temporary polygon data with your new attribute data
out.country<-merge(country_temp, sulferdioxide, by="name", all.x=TRUE)
out.country<-data.table(out.country)

#Set key to help ensure the manipulated data gets re-entered
#into the spatial polygon in the same order it came out
setkey(out.country, rn)
country_shapes@data<-out.country[row.names(country_shapes)]

#Decide palette to be used and determine number
#of desired colors

pal <- colorQuantile("YlGn",domain=NULL, n =5)

#Make your choropleth in leaflet
leaflet(data = country_shapes) %>%
      addProviderTiles("CartoDB.Positron") %>%
      addPolygons(fillColor = ~pal(`2005`),
                  fillOpacity = 0.8,
                  color = "black",
                  weight = 1,
                  popup=~paste(name, "<br>Rate:", country_shapes@data$`2005`) ) %>%
      addLegend("bottomleft",
                colors=brewer.pal(5,"YlGn"),
                labels=c("lowest","","","","highest"),
```

```
                  title="Relative Sulfer Dioxide Rates")
```

- Leaflet choropleth with ANIMATION in Shiny!

```
ui <- fluidPage(
  sliderInput(
    inputId="animation",
    label="Year",
    min=1975,
    max=2005,
    value =1975,
    step=1,
    animate=T),
  leafletOutput("mymap")
      )

server <- function(input, output) {
  output$mymap <- renderLeaflet({
    leaflet(data = country_shapes) %>%
      addProviderTiles("CartoDB.Positron") %>%
      addPolygons(fillColor = ~pal(eval(as.symbol(input$animation))),
                  fillOpacity = 0.8,
                  color = "black",
                  weight = 1,
                  popup=~paste(name, "<br>Rate:",
                               eval(as.symbol(input$animation)))) %>%
      addLegend("bottomleft",
                colors=brewer.pal(5,"YlGn"),
                labels=c("lowest","","","","highest"),
                title="Relative Sulfer Dioxide Rates")
      })
    }

shinyApp(ui, server)
```

- County-Level Choropleths in Leaflet

```
#Load in county-level geojson file
counties_shapes<-geojson_read("https://raw.githubusercontent.com/jgoodall/us-maps/master/geojson/co

#Subset the county-level data to include only
#The State FIP(s) of interest
#I looked up the FIP of California so that we could
#re-plot the pregnancy data

counties_shapes_CA<-counties_shapes[counties_shapes$STATEFP10=="06",]

#Import attribute data
CApregnancy <- read.csv("P:/QAC/qac251/course_materials/Code and Data/CApregnancy.csv")

#The row names contain valuable ID information. I do not want
#to lose this information so we will create a new variable
# called `rn` (I use this to denote rownames)

counties_shapes_CA$rn <- row.names(counties_shapes_CA)

#I create a temporary data.table to manipulate.
#It is recommended that you do not try to manipulate the data
#directly in a Large spatial polygon data.frame.
#Polygon ID information and labels get lost easily

counties_temp<-data.table(counties_shapes_CA@data)

#You may need to run the chunk of code below if your computer
#has encoding issues

#counties_temp$name <- iconv(counties_temp$name ,"WINDOWS-1252","UTF-8")

#As before, we need to make sure that the attribute data
#can be matched to the polygon data.
#The only difference I notice is the capitalization and id names.

#Let us have ID in both data sets equal to `ID`
counties_temp<-mutate(counties_temp, id=tolower(NAME10))
names(CApregnancy)[1]<-"id"
CApregnancy<-mutate(CApregnancy, id=tolower(id))

#Merge your temporary polygon data with your new attribute data
out.counties<-merge(counties_temp, CApregnancy, by=c("id"), all.x=TRUE)

#Again convert to data.table.
out.counties<-data.table(out.counties)

#Setting a key will help us to preserve original order of data
setkey(out.counties, rn)
counties_shapes_CA@data<-out.counties[row.names(counties_shapes_CA)]

#Setting color palette function
pal <- colorQuantile("RdPu",domain=NULL, n =5)
```

```
leaflet(data = counties_shapes_CA) %>%
  addProviderTiles("CartoDB.Positron") %>%
  addPolygons(fillColor = ~pal(X2009),
              fillOpacity = 0.8,
              color = "black",
              weight = 1,
              popup=~paste(id, "<br>Rate:", counties_shapes_CA@data$X2009) ) %>%
  addLegend("bottomleft",
            colors=brewer.pal(5,"RdPu"),
            labels=c("lowest","","","","highest"),
            title="Relative Teen Pregnancy Rates") %>%
  setView(-121.17, 37.9176, zoom = 5)
```

- Heatmaps in Leaflet

```
earth_quakes <- read.csv("P:/QAC/qac251/course_materials/Code and Data/earth_quakes.csv")


require(leaflet.extras)

earth_quakes2<-filter(earth_quakes, !is.na(longitude)&!is.na(latitude))

leaflet(data = earth_quakes2) %>%
  addProviderTiles("CartoDB.Positron") %>%
  addWebGLHeatmap(lng=~longitude,
                  lat=~latitude,
                  size=300000)

#Size is defaulted to meters. You can instead give a size in pixels.
```