UK Data Service

Creating Maps in R with Leaflet

Simon Parker
Data Liaison Manager – Cancer Research UK

Essex Summer School 2020
Day 2

Leaflet

- Leaflet is popular open-source JavaScript library for creating interactive maps. The leaflet package integrates the library with R.
- It is very flexible and allows users to create maps with data represented at specific points or as chloropleths (heatmaps).
- Functions like location search and popups can also be added.
- It can also be integrated into R Shiny allowing the construction of web applications with interactive maps.

  Example: https://crukcancerintelligence.shinyapps.io/UKDataMap/

Data

- In the first part of this session we are going to use data from the UK Police API. This API allows the public to access data about polices forces in the UK as well the locations of recorded crimes.
- There is a R package ukpolice that we are going to use to access the API.
- In the second part of the session we will use data from the 2011 UK census, which is available from the UK Data Service using the InFuse tool http://infuse.ukdataservice.ac.uk/.
- We will also be using a shapefile obtained from the Office for National Statistics Open Geography portal https://geoportal.statistics.gov.uk/

Producing our first map

1. Install and load the leaflet, dplyr and ukpolice packages.

1. Import crime data for East Anglia

```
crime <- ukc_street_crime(lat = c(52.971286, 52.538124, 52.538124, 52.971286),
                          lng = c(0.295886, 0.295886, 1.799542, 1.799542))
```

1. Prepare data for mapping

   Convert the latitude and longitude variables to numeric variables.
   Convert the category variable to a factor.

Producing our first map

```
crimemap <- leaflet(crime) %>%
              addTiles() %>%
            addMarkers(~longitude, ~latitude)
```

There are probably too many markers for this to be useful. There is a an easy fix to this though. Let's create the map again but this time we will specify that markers should be clustered.

```
crimemap2 <- leaflet(crime) %>%
                          addTiles() %>%
                          addMarkers(~longitude, ~latitude,
clusterOptions=markerClusterOptions())
```

Adding more information

Whilst we can now see the locations more easily, the map doesn't tell us a lot of information. We can use popup or labels to present information about the markers. Popups appear when the marker is clicked on, whereas labels appear whenever a user hovers over a marker.

We can use html to create the content of our popups and labels:

```
content <- paste("<b>Category:</b>", crime$category, "<br/>",
                 "<b>Location:</b>", crime$street_name, "<br/>",
                 "<b>Outcome:</b>", crime$outcome_status_category)


crimemap3 <- leaflet(crime) %>%
                        addTiles() %>%
                        addMarkers(~longitude, ~latitude, clusterOptions =
markerClusterOptions(),
                 popup = ~content)
```

Adding colour

We can use different markers on our maps as well as different basemaps (see http://leaflet-extras.github.io/leaflet-providers/preview/index.html for the basemaps).

In the next example we are going to use circle markers instead of the markers we've used previously. This will allow us to add colour to the markers.

Firstly though we need to define our colour palette which will be used for our markers. Leaflet gives us 4 functions to do so: *colorNumeric*, *colorBin*, *colorQuantile* for numeric variables, and *colorFactor* for categorical variables.

We have to specify the colours we wish to use and the variable that will be the domain. This will create function called 'pal' which can be used to generate colour.

```
pal <- colorFactor(rainbow(14), crime$category)
```

Adding colour

We can generate a map using the palette to colour the markers.

```
crimemap4 <- leaflet(crime) %>%
              addProviderTiles(providers$Esri.WorldGrayCanvas) %>%
              addCircleMarkers(~longitude, ~latitude, clusterOptions = markerClusterOptions(),
              popup = ~content, color = ~pal(category), opacity = 1, fillColor = ~pal(category),
              fillOpacity = 1)
```

Legends

Now we have colour to distinguish markers, it would be useful to have a legend to help readers identify what each colour means without having to click on the markers.

```
crimemap5 <- leaflet(crime) %>%
                addProviderTiles(providers$Esri.WorldGrayCanvas) %>%
                    addCircleMarkers(~longitude, ~latitude,
                clusterOptions = markerClusterOptions(),
                                popup = ~content,
                                color = ~pal(category), opacity = 1, fillColor = ~pal(category),
fillOpacity = 1) %>%
                                        addLegend("topright", pal = pal, values = crime$category,
                                        title = "Crime Category", opacity = 1)
```

Chloropleths

Now we will move on to chloropleths (sometimes called heatmaps). These are slightly more complicated as rather than displaying data at a particular point, we are now be showing data for a particular region. To make these work we will usually have to use a shapefile. A shapefile is actually a collection of files used in combination to map geographic areas.

The first step for this work is to install a few new packages to help us work with shapefiles.

```
install.packages("rgdal")
install.packages("sp")
install.packages("spdplyr")
```

Loading the shapefile

```
LAD <- readOGR(dsn = "./Path",
        layer = "Local_Authority_Districts__December_2011__Boundaries_EW_BGC",
        verbose = FALSE)
```

readOGR is part of the rgdal package.

As shapefiles are made up of multiple files we use *dsn* to specify the data source name, this is the folder in which the shapefiles can be found. The *layer* argument instructs R to pick up all files in the *dsn* that have the characters in their filename.

We can use *str* to inspect the shapefile to get a clearer view of the data inside the shapefile.

```
str(LAD, max.level = 3, list.len = 8)
```

Preparing the shapefile

Shapefiles need to have the map projection that Leaflet is expecting in order to display correctly. We can use the following code to set our shapefile to have the right projection:

```
PRO <- CRS('+proj=longlat +datum=WGS84 +ellps = WGS84 +towgs84 = 0,0,0')
LAD <- spTransform(LAD, PRO)
```

Now we can import our data. This data records the number of people who self-reported their general health in the 2011 census by local authority district :

```
GenHealth <- read_csv("GenHealth.csv")
```

Preparing our data

Exercise: Create a new variable ‘BadHealth’. This will present the percentage of people who reported bad or very bad health for the local authority district.

```
GenHealth <- GenHealth %>% mutate(Total = select(.,`General health : Very good health`:`General health : Very bad health` ) %>%
                                    rowSums(na.rm = TRUE))

GenHealth <- GenHealth %>% mutate(BadHealth = (((`General health : Bad health`+ `General health : Very bad health`) / Total) * 100)
```

Now we can merge our GenHealth dataframe with the LAD shapefile:

```
healthmap <- merge(LAD, GenHealth, by = "lad11cd")
```

Producing our chloropleth

We will use colorBin to produce a new colour palette for this map. Use quantile(healthmap$BadHealth) to help decide on suitable cuts.

```
pal2 <- colorBin("OrRd", bins = c(2, 4, 6, 8, 10), domain = healthmap$BadHealth)
```

Now we can produce our first chloropleth:

```
health.map <- leaflet(healthmap) %>%
                addProviderTiles(providers$Esri.WorldGrayCanvas) %>%
                addPolygons(weight = 0.5, color = "#d3d3d3", smoothFactor = 0.5,
                fillColor = ~pal2(BadHealth), fillOpacity = 0.8)
```

Improving our map

In the first version of our map our shapefile includes Wales although we don't have data for Wales. We can remove Welsh LADs from our shapefile to improve the appearance.

```
healthmap <- healthmap[ grep("E", healthmap$lad11cd) , ]
```

We can also add highlights to our map. This will change the appearance of an area when the mouse hovers over it.

```
health.map2 <- leaflet(healthmap) %>%
                    addProviderTiles(providers$Esri.WorldGrayCanvas) %>%
                    addPolygons(weight = 0.5, color = "#d3d3d3", smoothFactor = 0.5,
                    fillColor = ~pal2(BadHealth), fillOpacity = 0.8,
                    highlightOptions = highlightOptions(color = "white", weight = 2,
bringToFront =                          TRUE))
```

Exercise

1. Add a popup to the map we have produced. It should display the name of the Local Authority District and the percentage of people reporting bad health.

```
content2 <- paste("<b>Local Authority:</b>", healthmap$lad11nm, "<br/>",
                  "<b>Bad Health %:</b>", round(healthmap$BadHealth, 2))
```

2. Add a legend to the map. Create custom labels for the legend (for example: 2% – 4%).

```
labels <- c("2% - 4%", "4% - 6%", "6% - 8%", "8% - 10%")
labFormat = function(type, cuts, p) {paste0(labels)}
```

3. Using the leaflet.extras package add a reset button and a minimap to your map.

```
%>% addResetMapButton() %>% addMiniMap()
```