Data Visualisation using RShiny

Laurie Baker, Elaine Ferguson, Will Harvey and Rachel Steenson

Tanzania, August 2019

Course Overview

Day 1:

- 1.1 Getting to know your data
- 1.2 Data subsetting and summarising
- 1.3 Build exploratory plots
- 1.4 Building an interactive plot in RShiny

Day 2:

- 2.1 Introduction to mapping in R
- 2.2 Building a leaflet map in R
- **2.3 Build an interactive map in RShiny**

Day 3:

- 3.1 Review
- 3.2 Build your own apps!

Interacting with leaflet maps in Rshiny

Some interactive elements (e.g. pop-ups, zoom) are available in leaflet itself.

However, if we want to interactively change which data points or layers will be displayed on the map, we need to bring it into Shiny.

We can use many of the same widgets used to interact with regular plots to interact with leaflet maps!

A simple leaflet map to be made interactive

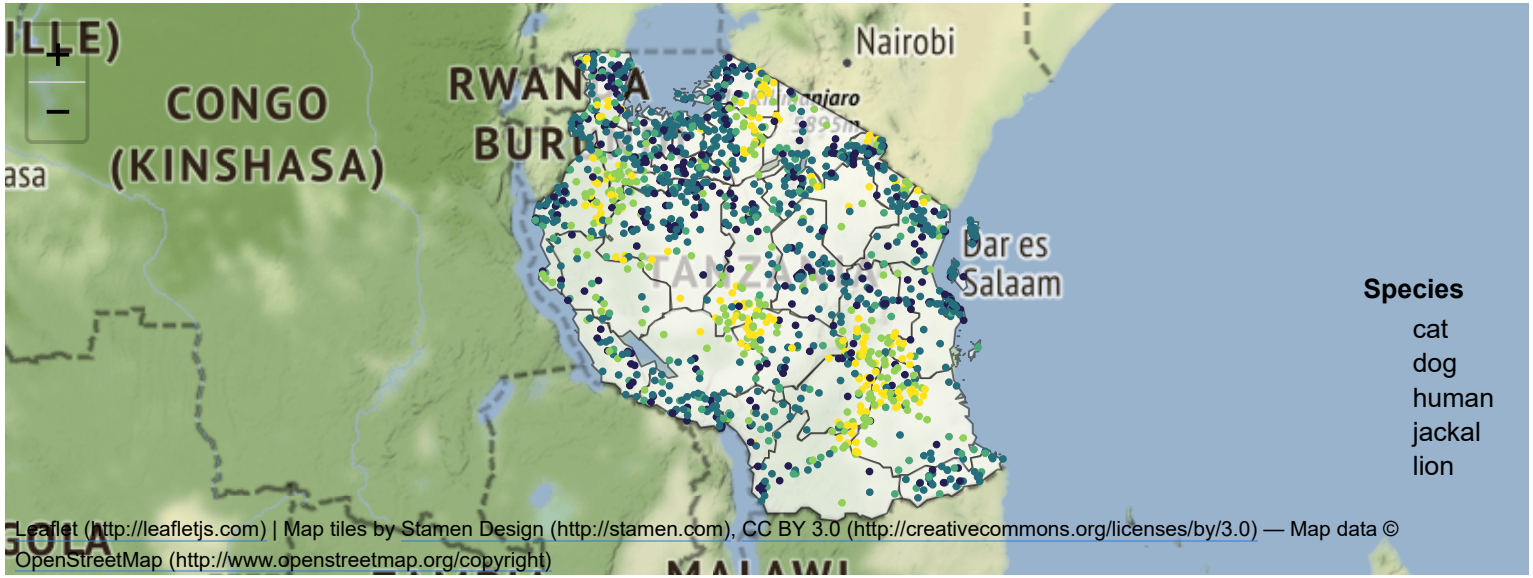
```
## Initialise map with tile. Set view and zoom.
m <- leaflet(width=800, height=300) %>%
  addProviderTiles("Stamen.Terrain") %>%
  setView(c(gCentroid(regions)@coords)[1], c(gCentroid(regions)@coords)[2],
          zoom = 5)
m
```

A simple leaflet map to be made interactive

```
## Add region shapefile
m <- m %>%
  addPolygons(data=regions,color="black",fillColor = "white",
              label=regions$Region_Nam, weight=1, fillOpacity=0.7)
m
```

A simple leaflet map to be made interactive

```
m %>%
  addCircles(data=leaflet_data,lng=~leaflet_data$x,lat=~leaflet_data$y,
             color = pal(leaflet_data[,"species"]), weight=2,
             opacity=1, fillOpacity=1, popup = popupInfo) %>%
  addLegend(position = "bottomright", title = "Species",
            pal = pal, values = leaflet_data[,"species"], opacity=1,
            labFormat = labelFormat(big.mark = ""))
```



What widgets could we add to make the map more interactive?

- `selectInput()` to select alternative variables to colour points by
- `sliderInput()` to choose date range of plotted points
- `checkboxGroupInput` to select which background polygon layers to display
- `pickerInput()` to choose a subset of the species to plot (from the **shinyWidgets** package)
- …

Outputting a leaflet map in Shiny

- `renderLeaflet()` is used in the **server** to create the leaflet map
- `leafletOutput()` is used in the **ui** to diplay the map

Create a map app with choice of variable to colour by

shinyUI Side Panel

```
selectInput(inputId="colourby", label="Colour Cases By:",
            choices = c("species","date"), selected="species")
```

shinyUI Main Panel

```
leafletOutput("mymap",width=1000,height=700)
```

Create a map app with choice of variable to colour by

shinyServer `reactive` section

```
pal <- reactive({
  colourby_col <- ifelse(input$colourby!="date",input$colourby,"date_decimal")
  if(input$colourby == "species"){
    colorFactor(palette, domain = sort(unique(leaflet_data[,colourby_col])))
  }else if(input$colourby == "date"){
    colorNumeric(palette, range(leaflet_data[,colourby_col]))
  }
})
```

- in an `if(){}` statement, the code inside the curly brackets is only evaluated if the conditions in the round bracket are met
- an `else if(){}` (or `else{}`) statement is only evaluated if the preceding `if(){}` statement is not

Create a map app with choice of variable to colour by

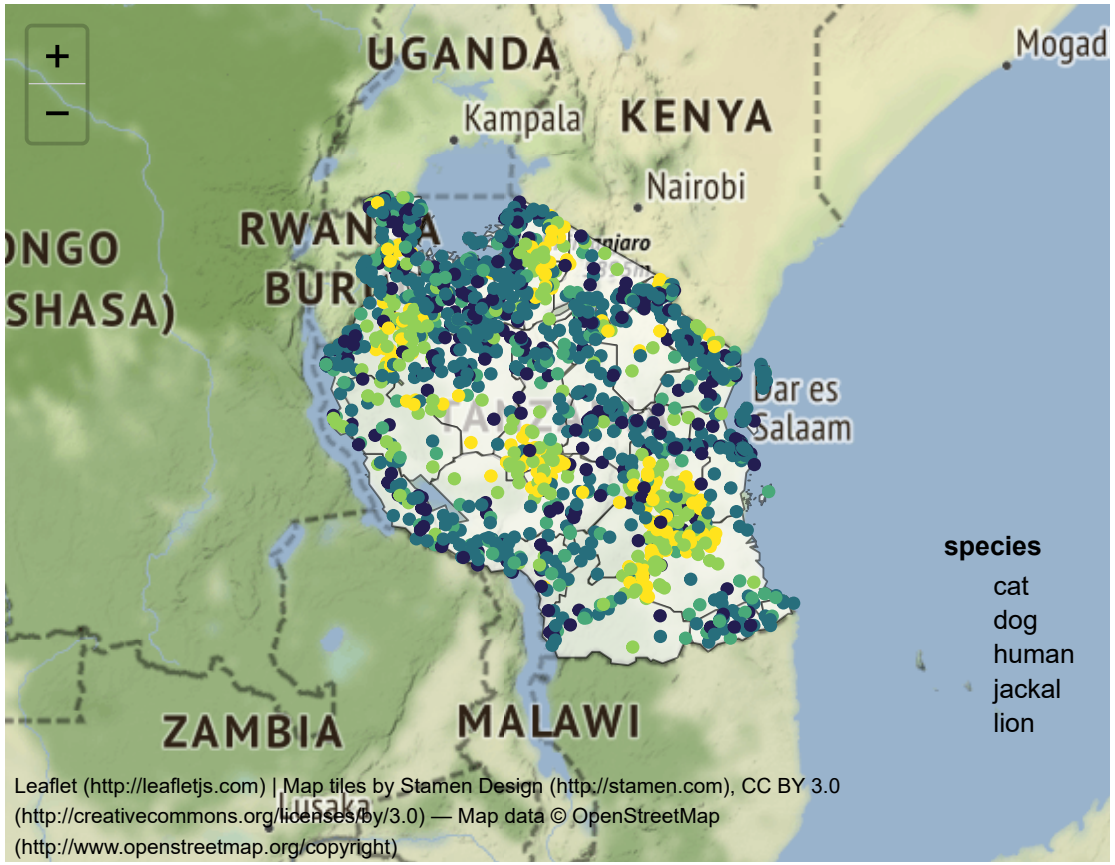
shinyServer `renderLeaflet()` section

```
output$mymap <- renderLeaflet({

  colourby_col <- ifelse(input$colourby!="date",input$colourby,"date_decimal")

  leaflet() %>%
    addProviderTiles("Stamen.Terrain") %>%
    setView(c(gCentroid(regions)@coords)[1], c(gCentroid(regions)@coords)[2], zoom = 6) %>%
    addPolygons(data=regions,color="black",fillColor = "white",
                label=regions$Region_Nam, weight=1, fillOpacity=0.7) %>%
    addCircles(data=leaflet_data,lng=~leaflet_data$x,lat=~leaflet_data$y,
               color = pal()(leaflet_data[,colourby_col]),
               opacity=1, fillOpacity=1, popup = popupInfo) %>%
    addLegend(position = "bottomright", title = input$colourby,
              pal = pal(), values = leaflet_data[,colourby_col], opacity=1,
              labFormat = labelFormat(big.mark = ""))
})
```

Create a map app with choice of variable to colour by

Colour Cases By:

species



Your turn

Complete section 2.3a of the handout.

Summary - section 2.3a

shinyUI Side Panel

```
selectInput(inputId="colourby", label="Colour Cases By:",
            choices = c("species","date","sex","age"),
            selected="species")
```

shinyServer `reactive` section

```
pal <- reactive({
  colourby_col <- ifelse(input$colourby!="date",input$colourby,"date_decimal")
  if(input$colourby %in% c("species","sex")){
    colorFactor(palette, domain = sort(unique(leaflet_data[,colourby_col])))
  }else if(input$colourby %in% c("date","age")){
    colorNumeric(palette, range(leaflet_data[,colourby_col]))
  }
})
```

The shinyWidgets package

- provides an extension of the widgets available in base R shiny
- many provide similar basic functions to those we've used already, but are more customisable or have added functionality
- these widgets can be explored in the shinyWidgets online gallery
- e.g. `pickerInput`

```
pickerInput(inputId = "species", label = "Species:",
            choices = sort(all_species), selected= all_species,
            options = list(`actions-box` = TRUE,`live-search` = TRUE), multiple = T),
```

Subsetting the data to be displayed

shinyServer new `reactive` sections

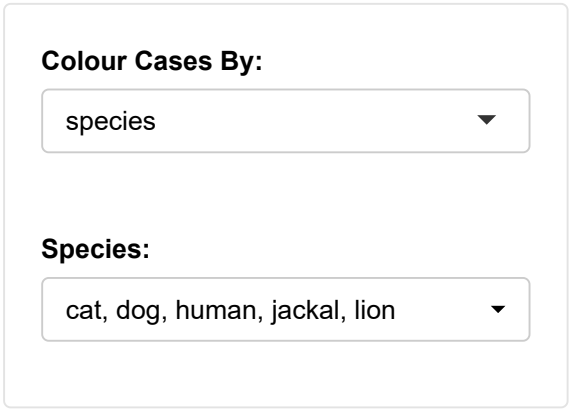
```
## Subset data based on inputs
leaflet_data_sub<- reactive({
  leaflet_data %>%
    filter(species %in% input$species)
})

## Create text pop-up information for each point in subsetted data
popupInfo <- reactive({
  paste("Date: ", leaflet_data_sub()$date, "<br>",
        "Species: ", leaflet_data_sub()$species, "<br>",
        "Age: ", leaflet_data_sub()$age, "<br>",
        "Sex: ", leaflet_data_sub()$sex, "<br>",
        sep = " ")
})
```

Subsetting the data to be displayed

shinyServer `renderLeaflet()` changes

```
addCircles(data=leaflet_data_sub(),lng=~leaflet_data_sub()$x,lat=~leaflet_data_sub()$y,
           color = pal()(leaflet_data_sub()[,colourby_col]),
           opacity=1, fillOpacity=1, popup = popupInfo()) %>%
addLegend(position = "bottomright", title = input$colourby,
          pal = pal(), values = leaflet_data_sub()[,colourby_col], opacity=1,
          labFormat = labelFormat(big.mark = ""))
})
```

Subsetting the data to be displayed



Your turn

Complete section 2.3b of the handout.

Summary - section 2.3b

Above shinyUI

```
all_regions <- unique(leaflet_data$region)
```

shinyUI Side Panel

```
pickerInput(inputId = "region", label = "Region:",
            sort(all_regions), selected= all_regions,
            options = list(`actions-box` = TRUE,`live-search` = TRUE), multiple = T)
```

shinyServer `reactive` section

```
leaflet_data_sub<- reactive({
  leaflet_data %>%
    filter(region %in% input$region)
})
```

Changing the displayed map layers

- As you add more map layers (shapefiles, rasters and points), your map can become overcrowded and take a long time to load
- Allowing the user to select the layers they're interested in can make your app faster and visually clearer

shinyUI Side Panel

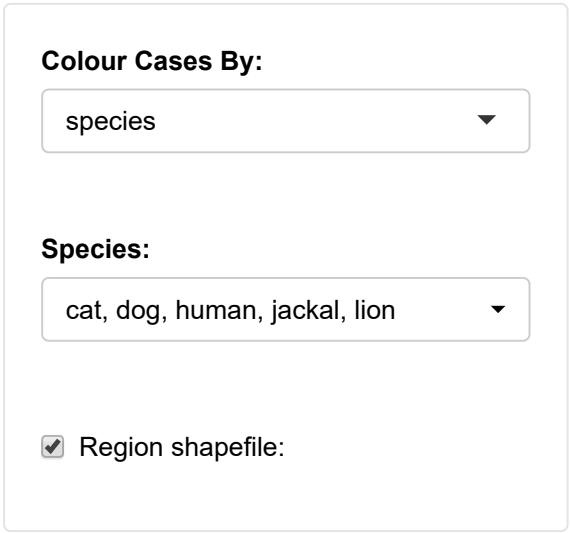
```
# Checkbox for choosing shapefiles to be displayed
checkboxInput("shapefile", label = "Region shapefile:", value = TRUE)
```

Changing the displayed map layers

shinyServer `leaflet` code

```
## Initialise map with tile. Set viewing window and initial zoom.
m <- leaflet() %>% addProviderTiles("Stamen.Terrain") %>%
  setView(c(gCentroid(regions)@coords)[1], c(gCentroid(regions)@coords)[2], zoom = 6)

## Add region shapefile if box is checked
if(input$shapefile == TRUE){
  m <- m %>% addPolygons(data=regions,color="black",fillColor = "white",
                         label=regions$Region_Nam, weight=1, fillOpacity=0.7)}

## Add coloured points and legend
colourby_col <- ifelse(input$colourby!="date",input$colourby,"date_decimal")
m %>% addCircles(data=leaflet_data_sub(),lng=~leaflet_data_sub()$x,lat=~leaflet_data_sub()$y,
                 color = pal()(leaflet_data_sub()[,colourby_col]),
                 opacity=1, fillOpacity=1, popup = popupInfo()) %>%
      addLegend(position = "bottomright", title = input$colourby,
                pal = pal(), values = leaflet_data_sub()[,colourby_col], opacity=1,
                labFormat = labelFormat(big.mark = ""))
```

Changing the displayed map layers



Your turn

Complete section 2.3c of the handout.

Summary - section 2.3c

shinyUI Side Panel

```
checkboxGroupInput("shapefiles", label = "Select background polygons:",
                   choices =  c("regions", "protected areas"),
                   selected = c("regions"))
```

above shinyServer

```
PAs <- readOGR("data/TZprotected_areas","TZprotected_areas")
```

shinyServer leaflet changes

```
if("regions" %in% input$shapefiles){
  m <- m %>%
    addPolygons(data=regions,color="black",fillColor = "white",
                label=regions$Region_Nam, weight=1, fillOpacity=0.7)}
if("protected areas" %in% input$shapefiles){
  m <- m %>%
    addPolygons(data=PAs,color="transparent",fillColor = "tomato",
                weight=1, fillOpacity=0.7)}
```

Subsetting the data by date

- Adding a widget to select a date range of interest lets the app user visualise how the data change in both space and time
- Complete **section 2.3d** of the handout.

Summary - section 2.3d

shinyUI Side Panel

```
sliderInput(inputId = "date", label = "Date:",
            min = min(leaflet_data$date), max =max(leaflet_data$date),
            value=c(min(leaflet_data$date), max(leaflet_data$date)),
            timeFormat="%b %Y")
```

shinyServer `reactive` function

```
 leaflet_data_sub<- reactive({
    leaflet_data %>%
      filter(date>input$date[1] & date<input$date[2] & species %in% input$species)
  })
```

Summary - section 2.3d