Shiny & Maps: a geo representation of earthquakes evolution with R

Andrea Melloncelli
andrea.melloncelli@quantide.com

Mariachiara Fortuna
mariachiara.fortuna@quantide.com

sponsored by QUANTIDE knowledge from data

ggmap package

ggmap: Spatial Visualization with ggplot2

by David Kahle and Hadley Wickham (R Journal, June 2013)

Spatial information of static maps

Sources:

- ➔ Google Maps
- ➔ OpenStreetMap
- ➔ Stamen Maps
- ➔ CloudMade Maps

+

Layered grammar of graphics of ggplot2

ggmap

Basic structure

```
my_map <- get_map(location = "Milano")

       ----------------------------------------------------------------------------

ggmap(my_map) +

       ----------------------------------------------------------------------------

geom_point(data = my_data,
    aes(x = lon, y = lat, colour = mag))
```

Get the map

Queries the Google Maps, OpenStreetMap, Stamen Maps or Naver Map servers for a map.

Plot the map

Plots the raster object produced by get_map()

Overlay data

Plots points or polygons and control their features through ggplot2

ggmap

get_map()

```
get_map(
 location = myLocation,
 zoom = "auto",
 maptype = c("terrain",
    "terrain-background", "satellite",
    "roadmap", "hybrid", ...),
 source = c("google",
            "osm",
            "stamen",
            "cloudmade"),
 color = c("color", "bw"),
 language = "en-EN")
```

See next slide :)

A value from 3 (continent) to 21 (building)

The map appearance

The map source, 4 available: GoogleMaps (default), OpenStreetMaps, Stamen, CloudMade (needs API)

Map color: color or black and white

Map language for GoogleMaps

ggmap

get_map(location =)

Location argument - 3 ways to define location

```
# Name or Address
myLocation <- “Milano”

# Lat/lon
myLocation <- c(lon = -95.3632715, lat = 29.7632836)

# Bounding box
myLocation <- c(min(lon), min(lat), max(lon), max(lat))
```

It’s possible to geocode an address! (query to Google Maps)

```
> geocode(“Milano”)
       lon      lat
  9.185924 45.46542
```

ggmap

ggmap()

ggplot2 structure, with some fixed layers

Fixed to the Mercator projection

x aesthetic fixed to longitude, y aesthetic fixed to latitude

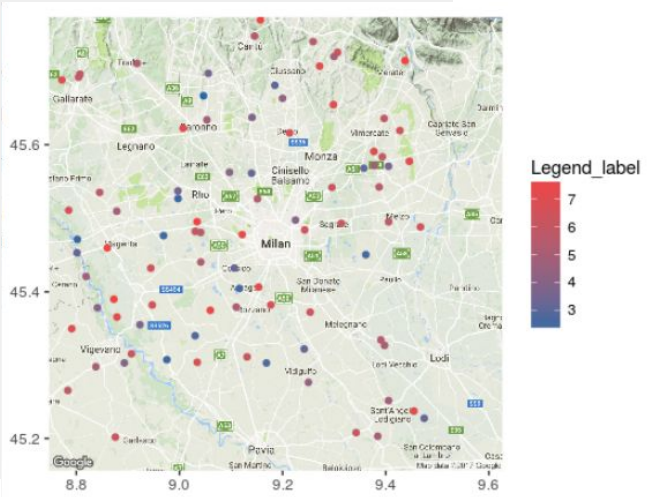
ggmap

Overlay data

```
my_map <- get_map(location = "Milano")

map_plot <- ggmap(myMap) +
     geom_point(data = myData, alpha = 0.5,
          aes(x = longitude, y = latitude, colour = mag)) +
     scale_colour_gradient("Legend_label",
          low = "#1E6AA8", high = "#F54242") +
     labs(x = NULL, y = NULL)

print(map_plot)
```



Shiny App Structure

```
# global ----------------------------------------
# This code is run once

    library(shiny)
    ....

# ui --------------------------------------------
# Draw graphical user interface of the app
    ui <- fluidPage(
       ....
    )

# server ----------------------------------------
# Logic behind the graphical interface
    server <- function(input, output, session){
      ....
    }
    shinyApp(ui = ui, server = server)
```

app.R :

1. global:

   initialization

2. ui:

   User Interface

3. server

   Server logic

Single file shiny app

Global

```
# global

# This code is run once

    library(shiny)

    source("other-code.R")
```

Goals:

1. load necessary libraries
2. execute some R code in other files

Single file shiny app

Shiny App Structure

```
# ui

# Draw graphical user interface of
the app
    ui <- fluidPage(
        ....
    )
```

Goals:

1. setup the aspect of the web app
2. collect input through widget objects
3. show output through output objects

Single file shiny app

Server

```
# server

# Logic behind the graphical
interface
    server <- function(input,
output, session){
      ....
    }
    shinyApp(ui = ui, server =
server)
```

Goals:

1. connect inputs to output objects
2. update output if input is changed

Single file shiny app

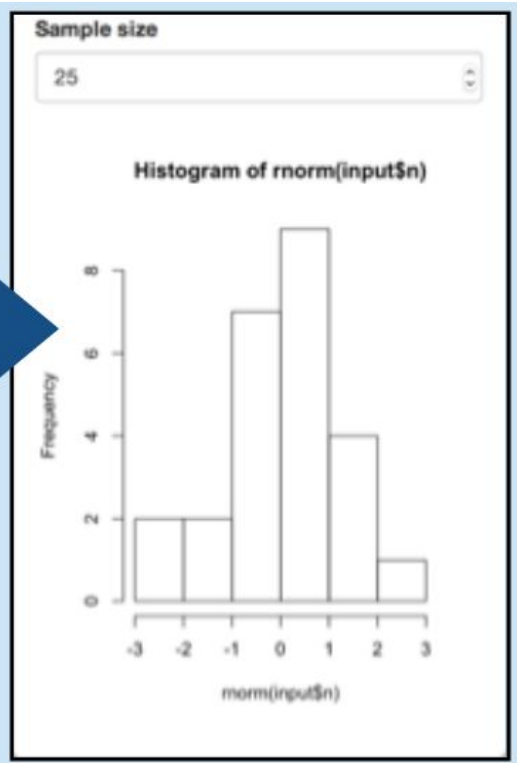
Shiny App Example

```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}

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



Cheat sheet

```
library(shiny)
ui <-
    fluidPage(
        numericInput(
            inputId = "num",
            label = "Sample size",
            value = 25),
        plotOutput(
            outputId = "hist"
        )
    )
server <-
    function(input, output) {
        output$hist   <- renderPlot({
            hist(rnorm(
                input$num
            ))
        })
    }
shinyApp(ui = ui, server = server)
```

Input Output cycle

```
library(shiny)
ui <-
     fluidPage(
          numericInput(
               inputId = "num",
               label = "Sample size",
               value = 25),
          plotOutput(
               outputId = "hist"
          )
     )
server <-
     function(input, output) {
          output$hist    <- renderPlot({
               hist(rnorm(
                    input$num
               ))
          })
     }
shinyApp(ui = ui, server = server)
```

[Single file shiny app]

1. ui:

   Variables are strings

2. server:

   Variables are object of `output` or `input` lists

Input Output cycle

```
library(shiny)
ui <-
     fluidPage(
          numericInput(
               inputId = "num",
               label = "Sample size",
               value = 25),
          plotOutput(
               outputId = "hist"
          )
     )
server <-
     function(input, output) {
          output$hist    <- renderPlot({
               hist(rnorm(
                    input$num
               ))
          })
     }
shinyApp(ui = ui, server = server)
```

Single file shiny app

Run:

1. ui : the user select a value for "Sample size"
2. server: input$num is modified
3. server: hist(rnorm(...)) is run again to update output$hist
4. ui: the hist is showed in the web inteface

Function substutution

```
library(shiny)

plot_fun <- function(number) {
   hist(rnorm(
     number
   ))
}

ui <-
     fluidPage(
          # ...
          )
     )

server <-
     function(input, output) {
          output$hist     <- renderPlot({
               plot_fun( input$num )
          })
     }
shinyApp(ui = ui, server = server)
```

Single file shiny app

Run:

1. ui : the user select a value for "Sample size"
2. server: input$num is modified
3. server: hist(rnorm(...)) is run again to update output$hist
4. ui: the hist is showed in the web inteface

Shiny & Maps: a geo representation of earthquakes evolution with R

Now hands on R code!

R-Labber groups

Groups:

1. shiny app
2. ggmap
3. ggplot

Single file shiny app

Group's common ground

```
plot_map <-
   function(
     tbl_eq,
     map_eq,
     period,
     mag,
     ...
   ) {
       # … code …
       return(ggplot_object)
     }
```

Function signature:

1. function name
2. list of arguments and their type
3. return type

Single file shiny app

Group's common ground

```
library(shiny)

plot_map <- function(tbl_eq, map_eq, period, mag, ...){

       # … code …
       return(ggplot_object)
}

ui <-
     fluidPage(
          # ...
          plotOutput(outputId = "map")
          # ... )

server <-
     function(input, output) {
          output$map      <- renderPlot({
             plot_map(tbl_eq, map_eq, period, mag, ...)
          })
     }

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

Input handling

Plot_map

```
plot_map <-
   function(
     tbl_eq,
     map_eq,
     period,
     mag,
     ...
   ) {
       # … code …
       return(ggplot_object)
     }
```

Arguments:

1. tbl_eq(data.frame): table of earthquakes
2. map_eq(ggmap) background map object
3. period(int): the year
4. mag(numeric): the magnitude threashold
5. … : other args

Single file shiny app