Technical Document

HealthWebMapper2.0

Project conducted by:
San Diego State University
5500 Campanile Drive
San Diego, CA 92182-4493

Report prepared by:
Haihong Huang
Department of Geography, San Diego State University

May 2019

Abstract

HealthWebMapper 2.0 is a web mapping application designed for visualizing cancer disparities problems in San Diego Sub-Regional Areas (SRAs). It was developed by R Shiny web development framework. HealthWebMapper2.0 allows users to upload their cancer data and socioeconomic data in San Diego sub-regional areas and visualize them in side-by-side synchronous interactive maps. Users can also view their input data table and get statistical summary, or conduct correlation (Pearson's r) analysis between selected cancer data and socioeconomic factor as well as spatial autocorrelation analysis for cancer data. This technical document contains 7 sections, aiming at helping future developers understand the mechanism behind HealthWebMapper2.0:

(1) Introduction to Shinyapps

(2) Input elements of HealthWebMapper2.0

(3) Output elements of HealthWebMapper2.0

(4) Non-reactive elements of HealthWebMapper2.0

(5)The UI functions of HealthWebMapper2.0

(6) The server function of HealthWebMapper2.0

(7) Sharing HealthWebMapper2.0

Part 1 Introduction to Shinyapps

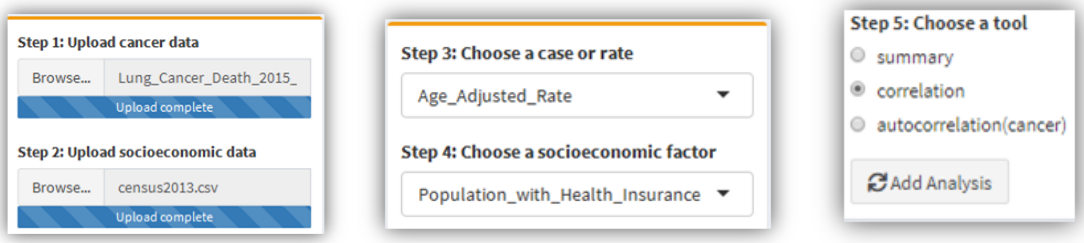
"Shiny is an R package that makes it easy to build interactive web applications (apps) straight from R programming language" (https://shiny.rstudio.com/). To build a web application with Shiny you need to learn R (https://www.rstudio.com/)

Every Shiny app is maintained by a computer running R. A Shiny app composes of two major parts: "ui" and "server". ui is where you add input and output elements to your apps as arguments. Server is the place you assemble inputs into outputs. To learn R shiny, please refer to https://shiny.rstudio.com/tutorial/.

This technical document focuses on explaining how HealthWebMapper2.0 was built with R shiny: its reactive input elements, reactive output elements, non-reactive elements, ui function, server function and how to share shinyapps.

Part 2 Input elements of HealthWebMapper2.0

HealthWebMapper2.0 has three major groups of reactive input elements, i.e. two file inputs, two drop-down menus and a radio button as showed in the figure below:

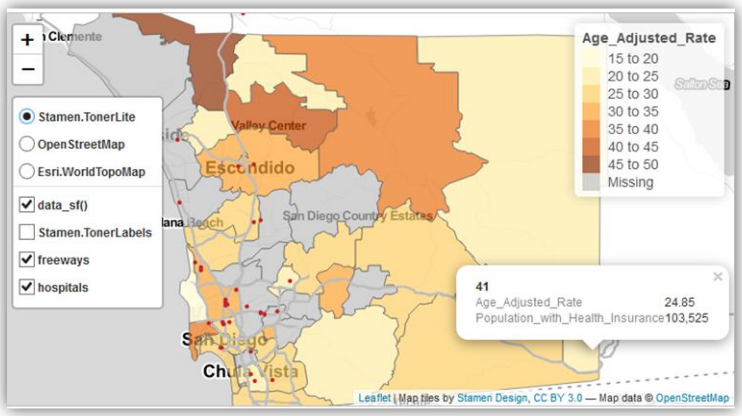


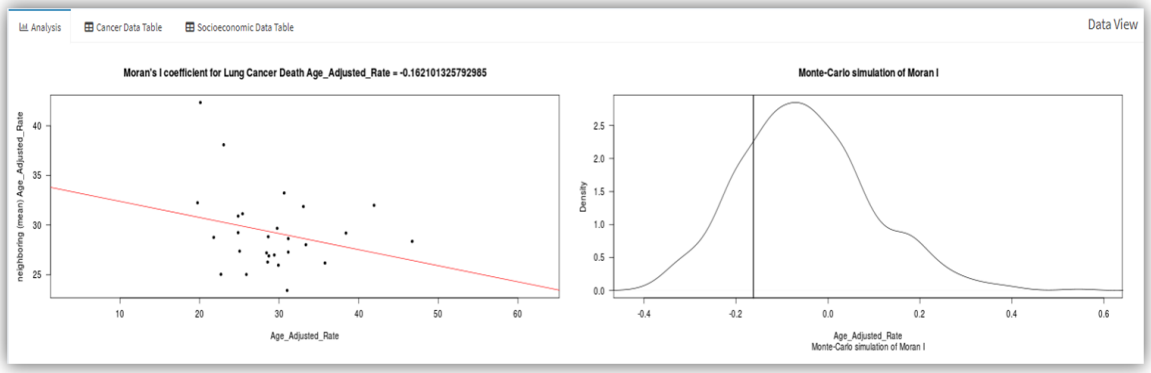
Their inputIds are "file1", "file2", "VARL", "VARR", "tool". The input functions for these three groups are *fileInput(), selectInput(), radioButtons*() (https://shiny.rstudio.com/gallery/widget-gallery.html).

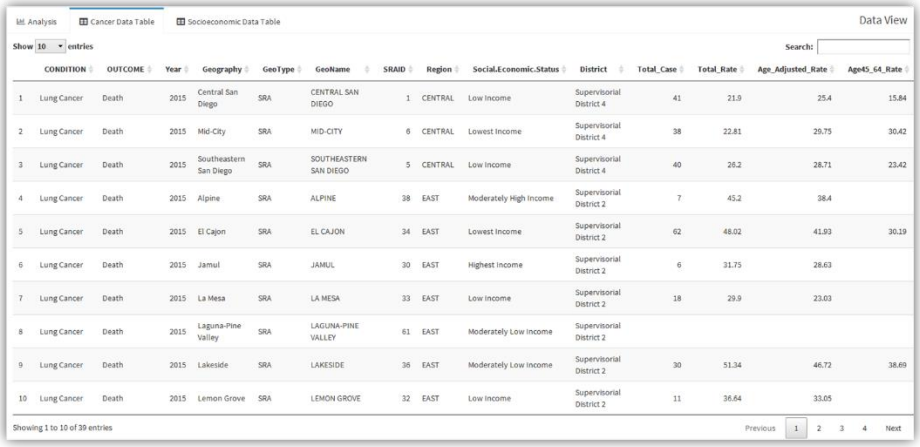
```
fluidRow(
  column(width = 2,
         box(width = NULL, status = "warning",
             fileInput("file1",
                       label = "Step 1: Upload cancer data",
                       accept = c("text/csv","text/comma-separated-values, text/plain",".csv")),
             fileInput("file2",
                       label = "Step 2: Upload socioeconomic data",
                       accept = c("text/csv","text/comma-separated-values, text/plain",".csv"))
             ),
         box(width = NULL,status = "warning",
             selectInput("VARL",label = "Step 3: Choose a case or rate", choices = c()),
             selectInput("VARR",label = "Step 4: Choose a socioeconomic factor", choices = c()),
             radioButtons('tool', label = "Step 5: Choose a tool", c('summary','correlation', 'autocorrelation(cancer)')),
             actionButton("add", label = "Add Analysis", icon = icon("refresh"))
             ),
```

Part 3 Output elements of HealthWebMapper2.0

There are three major groups of reactive output elements map, analysis and tables.







Their outputIds are “map”, “Analysis”, “table 1”, “table 2”. The output functions are *uiOutput(), DT::dataTableOutput()*

```
column(width = 10,
       box(title="Maps", status = "primary", height = "480px", width=NULL,
           uiOutput(outputId = "map")
         ),
       tabBox(title="Data View", id= "tabset", height = "1000px", width= NULL,
              tabPanel(title = "Analysis", icon = icon("bar-chart-o"),div(id = "placeholder")),
              tabPanel(title = "Cancer Data Table",icon = icon("table"),DT::dataTableOutput(outputId = "table1") ),
              tabPanel(title ="Socioeconomic Data Table",icon = icon("table"),DT::dataTableOutput(outputId = "table2"))
              )
       )
      )
```

Part 4 Non-reactive elements of HealthWebMapper2.0

Non-reactive elements in HealthWebMapper2.0 are website header with links and disclaimer:

```
header <- dashboardHeader(
  title = "HealthWebMapper",
  titleWidth = 300,
  tags$li(class="dropdown", tags$a(href="http://humandynamics.sdsu.edu/", tags$img(src="logo_HDMA.png", heigh='250', width='300'), target ="_blank")),
  tags$li(class="dropdown", tags$a(href="https://humandynamics.sdsu.edu/demo-data.html", icon("table", "fa-3x"),"Demo Data", target ="_blank")),
  tags$li(class="dropdown", tags$a(href="https://github.com/HDMA-SDSU/HealthWebMapper2", icon("github", "fa-3x"), "Tutorial", target ="_blank"))
)
```

```
box(width = NULL,status = "warning", title = "Disclaimer",
    uiOutput("disclaimer")
   )
```

Part 5 The UI function of HealthWebMapper2.0

In user interface (ui) of HealthWebMapper2.0, “dashboardPage” (https://rstudio.github.io/shinydashboard/) was adopted. It needs three major arguments: header, dashboardSidebar(disabled) and body.

```
ui <- dashboardPage(
  header,
  dashboardSidebar(disable = TRUE),
  body
  #skin = "red"
)
```

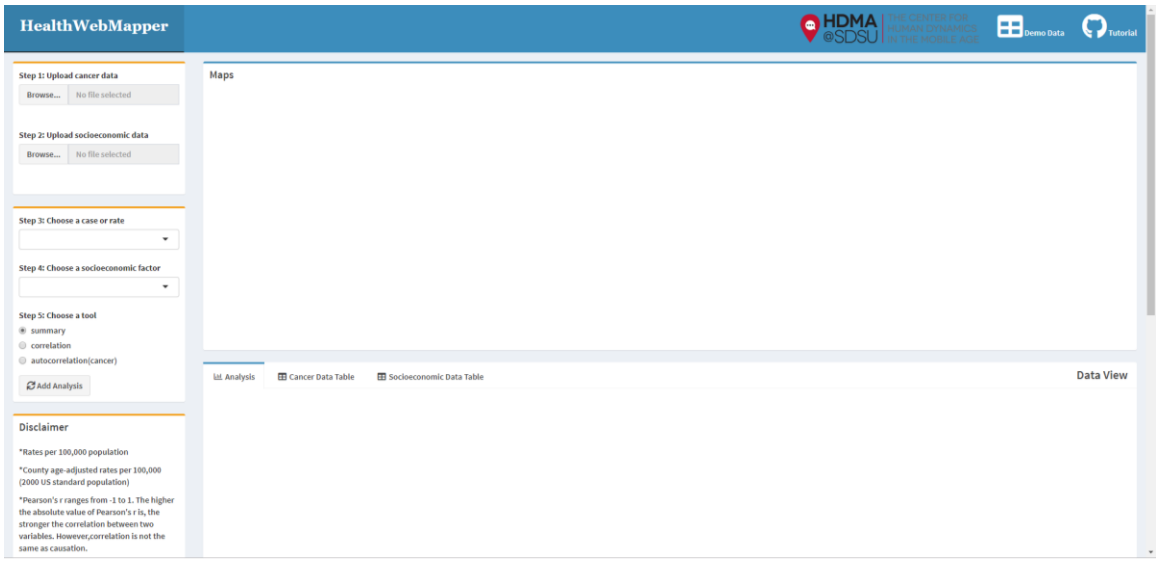
To define header argument, we define title, title width and adds three links with icons

```
header <- dashboardHeader(
  title = "HealthWebMapper",
  titleWidth = 300,
  tags$li(class="dropdown", tags$a(href="http://humandynamics.sdsu.edu/", tags$img(src="logo_HDMA.png", heigh='250', width='300'), target ="_blank")),
  tags$li(class="dropdown", tags$a(href="https://humandynamics.sdsu.edu/demo-data.html", icon("table", "fa-3x"),"Demo Data", target ="_blank")),
  tags$li(class="dropdown", tags$a(href="https://github.com/HDMA-SDSU/HealthWebMapper2", icon("github", "fa-3x"), "Tutorial", target ="_blank"))
)
```

The body is the place we put all the input, output and non-reactive elements. Within the body, *fluidRow(), column(), box()and tabbox()* was used to defined the shape, position and size of each elements. (https://shiny.rstudio.com/reference/shiny/latest/fluidPage.html; https://rstudio.github.io/shinydashboard/structure.html)

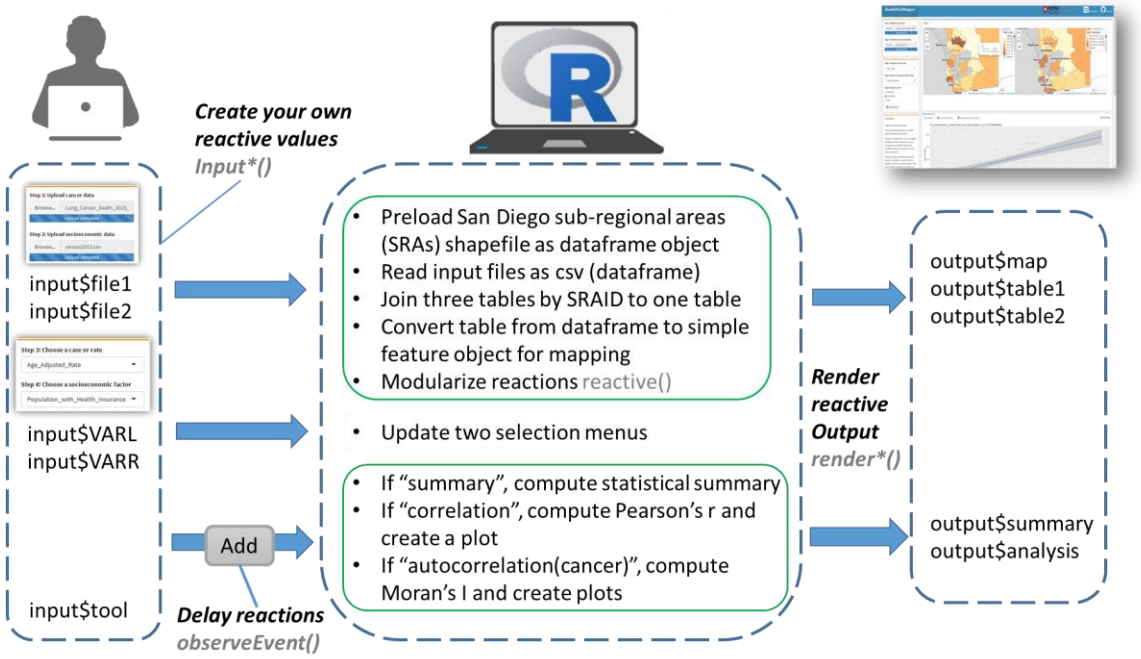
```
body <- dashboardBody(
  tags$head(
    tags$link(rel = "stylesheet", type = "text/css", href = "custom.css"),
    tags$style(type = "text/css", "#map {height: calc(100vh - 80px) !important;}")
  ),
  fluidRow(
    column(width = 2,
           box(width = NULL, status = "warning",
               fileInput("file1",
                         label = "Step 1: Upload cancer data",
                         accept = c("text/csv","text/comma-separated-values, text/plain",".csv")),
               fileInput("file2",
                         label = "Step 2: Upload socioeconomic data",
                         accept = c("text/csv","text/comma-separated-values, text/plain",".csv"))
               ),
           box(width = NULL,status = "warning",
               selectInput("VARL",label = "Step 3: Choose a case or rate", choices = c()),
               selectInput("VARR",label = "Step 4: Choose a socioeconomic factor", choices = c()),
               radioButtons('tool', label = "Step 5: Choose a tool", c('summary','correlation', 'autocorrelation(cancer)')),
               actionButton("add", label = "Add Analysis", icon = icon("refresh"))
               ),
           box(width = NULL,status = "warning", title = "Disclaimer",
               uiOutput("disclaimer")
              )
           ),
    column(width = 10,
           box(title="Maps", status = "primary", height = "480px", width=NULL,
               uiOutput(outputId = "map")
              ),
           tabBox(title="Data View", id= "tabset", height = "1000px", width= NULL,
                  tabPanel(title = "Analysis", icon = icon("bar-chart-o"),div(id = "placeholder")),
                  tabPanel(title = "Cancer Data Table",icon = icon("table"),DT::dataTableOutput(outputId = "table1") ),
                  tabPanel(title ="Socioeconomic Data Table",icon = icon("table"),DT::dataTableOutput(outputId = "table2"))
                  )
           )
         )
)
```

The final user interface looks like this:



Part 6 The server function of HealthWebMapper2.0

In the server function, inputs was assembled to outputs according to following diagram:



1. At the beginning of the code, preload San Diego sub-regional areas (polygon), freeways (polylines) and hospitals (points) shapefiles and extract necessary attributes SRAID and geometry

```
# import shpfiles once
SD_SRA_raw <- st_read("polygon/polygon.shp")
freeways<- st_read("polygon/freeways_WGS84.shp")
hospitals <- st_read("polygon/hospitals_WGS84.shp")

sd_sra <- data.frame(
  SRAID = as.numeric(SD_SRA_raw$SRA),
  geometry = SD_SRA_raw$geometry
)
```

2. In server function, read input files as csv (saved as dataframe objects) and modularize reactions

```
# two reactive objects of inputfiles
fileinputs1 <- reactive(
  {
    req(input$file1)
    read.csv(input$file1$datapath,
             header = TRUE)
  }
)

fileinputs2 <- reactive(
  {
    req(input$file2)
    read.csv(input$file2$datapath,
             header = TRUE)
  }
)
```

3. join three table by SRAID to one table and convert table from dataframe to simple feature object for mapping and modularize reaction

```
# join two input tables by SRAID
data_sf <- reactive({
  st_as_sf(left_join(left_join(sd_sra, fileinputs1(), by = 'SRAID'),left_join(sd_sra, fileinputs2(), by = 'SRAID'), by = 'SRAID'))
}
)
```

4. update selectinput based on input data

```
  # update SelectInput with reactive objects
  observe({
    #update left selection
    updateSelectInput(session, "VARL", choices = names(fileinputs1()), selected = "SRAID")
    #update right selection
    updateSelectInput(session, "VARR", choices = names(fileinputs2()), selected = "SRAID")
  })
```

5. create map with tmap package(https://cran.r-project.org/web/packages/tmap/vignettes/tmap-getstarted.html) and render maps using *renderUI()*

```
#Output side-by-side choropleth maps

output$map <- renderUI({

#    opts <- tmap_options(basemaps = c(Blackwhite = "Stamen.TonerLite", OpenStreetMap = "OpenStreetMap", Imagery = "Esri.WorldImagery"))

   tm <- tm_basemap(c("Stamen.TonerLite","OpenStreetMap", "Esri.WorldImagery"))+
         tm_shape(data_sf()) + tm_polygons(c(input$VARL, input$VARR), alpha = 0.7) + tm_facets(sync = TRUE, ncol = 2) +
         tm_tiles(leaflet::providers$Stamen.TonerLabels) +
         tm_shape(freeways)+tm_lines("grey", lwd = 3)+
         tm_shape(hospitals)+tm_dots(col = "red", shape = 3)

   tmap_leaflet(tm)


  })
```

6. Reaction button ‘Add” will observe events, once users select one of the three analysis tool and click button “Add”, the sever will process the data and generate corresponding results and display them in the analysis output elements

```
observeEvent(input$add, {
  id <- paste0(input$tool, input$add)

  insertUI(selector = "#placeholder",
           where = "afterEnd",
           ui = switch(input$tool,
               'summary' = verbatimTextOutput(id),
               'correlation' = plotOutput(id),
               'autocorrelation(cancer)' = plotOutput(id)
               )
           )

  dataset <- as.data.frame(data_sf())
  # preprocess data for correlation computation
  subset <- dataset[, c(input$VARL, input$VARR)]
  new_data <- subset[complete.cases(subset),]
  # preprocess data for autocorrelation computation
  subset1 <- dataset[, c(input$VARL, "geometry.x", "geometry.y")]
  new_data1 <- st_as_sf(na.omit(subset1))
  new_data2 <- as.data.frame(new_data1)
```

If select tool “correlation”, compute Pearson’s r coefficient and generate a scatter plot:

```
output[[id]] <-
  if (input$tool == 'correlation') renderPlot({
    tryCatch(
      {
        cor <- cor.test(new_data[,1], new_data[,2], method = "pearson", conf.level = 0.95)
      },
      error = function(e){
        stop(safeError("Check if the values of your selected variables are both numeric"))
      }
    )
    scatter_plot <- ggplot(isolate(new_data), aes_string(x=names(new_data)[1], y=names(new_data)[2]))
    scatter_plot + geom_point() + geom_smooth(method="lm")+ ggtitle(paste0("The correlation(Pearson's r) between ",isolate(input$VARL)," and ",isolate(input$VARR),

  })
```

Else if select tool “summary”:

```
else if (input$tool == 'summary') renderPrint({summary(dataset)})
```

Else if select tool “autocorrelation(cancer)”

```
else if (input$tool == 'autocorrelation(cancer)') renderPlot({
  # autocorrelation
  tryCatch(
    {
      isolate(new_data1)
      isolate(new_data2)
      nb <- poly2nb(new_data1, queen=TRUE)
    #assign weights to each neighboring polygon. In our case, each neighboring polygon will be assigned equal weight (style="W").
      lw <- nb2listw(nb, style="W", zero.policy=F)
    },
    error = function(e) {
      # return a safeErrorif too many missing data cause empty neighbor found
      stop(safeError("Unable to perform spatial autocorrelation analysis when there are too many missing data in the input cancer dataset
    }
  )

  #compute the average neighbor varibale value for each polygon. These values are often referred to as spatially lagged values.
  Inc.lag <- lag.listw(lw, new_data2[,1])
  # Create a regression model
  M <- lm(Inc.lag ~ new_data2[,1])
  MC<- moran.mc(new_data2[,1], lw, nsim=599)
  # Plot the data
  attach(mtcars)
  par(mfrow=c(1,2))
  plot(Inc.lag~ new_data2[,1], pch=20, asp=1, las=1, xlab= names(new_data2[1]), ylab= paste0("neighboring (mean) ",names(new_data2[1])),
  abline(lm(Inc.lag ~ new_data2[,1]), col="red")
  plot(MC, main= "Monte-Carlo simulation of Moran I", las=1,xlab= names(new_data2[1]))

  })
```

7. render data table to table output

```
# render table1
output$table1 <- DT::renderDataTable(DT::datatable({
  fileinputs1()
}))

# render table1
output$table2 <- DT::renderDataTable(DT::datatable({
  fileinputs2()
}))
```

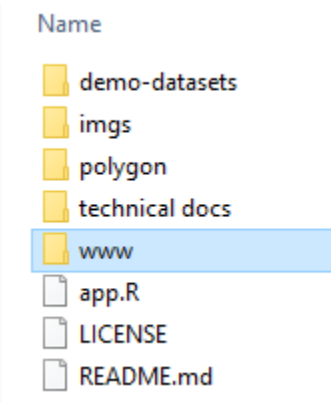
8. render contents into disclaimer output

```
# Disclaimer
output$disclaimer <- renderUI(
  {
    div(
    p("*Rates per 100,000 population"),
    p("*County age-adjusted rates per 100,000 (2000 US standard population)"),
    p("*Pearson's r ranges from -1 to 1. The higher the absolute value of Pears
    p("*Please interpret with these results with caution - correlation, is not
    p("*Moran's I ranges from -1 to 1. -1 is perfect clustering of dissimilar
    p("*In this tool, we accepct any contiguous polygon that shares at least o
    p("*In a Monte Carlo test, the attribute values are randomly assigned to p
    )


  }
)
}
```

Part 7 Sharing HealthWebMapper2.0

1. Save app.R and supporting www folder into one directory like below:



2. You can choose share and maintain your shinyapps in shinyapps.io (https://www.shinyapps.io/) or set up a Shiny server (https://www.rstudio.com/products/shiny/shiny-server/). Both ways has free version and paid version. Procedures for setting up server were available online.