An introduction to Shiny apps for health and demographic research

José Manuel Aburto
jmaburto@health.sdu.dk

1 Building an easy shiny app

1.1 Building the user interface

Open the R-script ui.R. Load the shiny package.

```
library(shiny)
```

Now start an easy example of user interface. The `ui.R` file registers a user interface with Shiny. `fluidPage` creates fluid page layouts. A fluid page layout consists of rows which in turn include columns. Rows exist to make sure their elements appear on the same line (if the browser has adequate width). Columns exist for the purpose of defining how much horizontal space elements should occupy. Fluid pages scale their components in real time to fill all available browser width. `tabsetPanel` creates a tabset that contains tabPanel elements. Tabsets are useful for dividing output into multiple independently viewable sections.

To create a sidebar and a main area in the layout we use `sidebarLayout`, we add a set of radio buttons to select from an item of a list with `radioButtons`.

```
fluidPage(
  titlePanel('My shiny app'),
  tabsetPanel(
    tabPanel("An example with histograms",
             sidebarLayout(
               sidebarPanel(
                 radioButtons(inputId='n',label= 'No. of observations',
                              list(10,100,1000,10000)),
                 br(),
                 p('Select the No. of observations for the histogram')),
                 mainPanel(
                   tabPanel('Histogram', plotOutput('histogram'))
                 )))
))
```

1.2 Building the server functions

Now we need a `server.R` file where we will allocate the outputs that we want. In this case, want to show a histogram with the input called `n`. Open the `server.R` file.

Load the packages that we will need

```
library(shiny)
library(ggplot2)
library(plotly)
library(DT)
library(RColorBrewer)
```

`shinyServer` defines the server-side logic of the Shiny application. This generally involves creating functions that map user inputs to various kinds of output. `renderPlot` Renders a reactive plot that is suitable for assigning to an output slot.

```
  function(input,output){

    output$histogram <-  renderPlot({
      df  <- rnorm(input$n)
      df  <- data.frame(df)

      fig <- ggplot(data = df, aes(x = df))+
                geom_histogram(colour = 'black', fill = 'white',
                               binwidth = 0.2)

      fig
    })
  }
```

Now we are ready to see our app. You can do so with the play button on Rstudio, or simply running

```
runApp()
```

We will create a more interactive example with the package `plotly`. This package easily translates `ggplot2` graphs to an interactive web-based version and/or create custom web-based visualizations directly from R. For this we will add a new `tabPanel` to our UI (inside `tabsetPanel`). Note that `numericInput` creates an input control for entry of numeric values.

```
      tabPanel("An example with interactive histograms",
               sidebarLayout(
                 sidebarPanel(
                   numericInput(inputId='n2',label= 'No. of observations',
                                value = 1000,
                                min = 10,max = 100000000,step = 15),
                   br(),
                   p('Select the No. of observations for the histogram')
                 ),
                 mainPanel(
                   tabPanel('Histogram', plotlyOutput('histogram2'))
                 )))
```

Now we add out new `histogram2` to the server.R file. `ggplotly` converts a `ggplot2::ggplot()` object to a plotly object.

```
output$histogram2 <- renderPlotly({

      df  <- rnorm(input$n2)
      df  <- data.frame(df)

      fig <- ggplot(df, aes(x = df)) +
        geom_histogram(colour = "black", fill = "white",binwidth = 0.2)

      ggplotly(fig,tooltip = c('count'))

    })
```

2 Life years lost

The aim of this exercise is to illustrate how to graph LYL with data on Saudi Arabia from the Global Burden of Disease project and United Nations. First, we add a panel in the `ui.R` file where the new subsection on LYL will be in our app. In the sidebar we will give the option to the user to select the period and sex that will be shown. In addition, we will print a table in a subpanel with the total LYL.

```
tabPanel("Life years lost for Saudi Arabia",
                sidebarPanel(
                  p('Select a period to show LYL.'),
                  selectInput( 'Period.Ind','Period for LYL',
                               c("2000-2005","2005-2010","2010-2015"),
                               selected = "2000-2005"),
                  p('Select the sex.'),
                  selectInput( 'Sex.Ind','Sex',c("Female","Male"),
                               selected = "Female")
                )
         ,
                mainPanel(
                  tabsetPanel(
                    tabPanel('Plot',
                             plotlyOutput('LYL.plot',width = '100%')
                    ),
                    tabPanel('Table',
                             dataTableOutput('LYL.table'))
                  ))
         )
```

Now, we load the data in our server function

```
load('ResultsLYL.RData')
```

These data contain two objects. Results.LYL has information on age, sex, cause, LYL, periods, and some graph parameters (ymin, ymax). Table LYL contains the contribution of causes of death to the LYL. First we construct the graph by adding a new instruction in our server file.

```
output$LYL.plot <- renderPlotly({

      Period1  <- input$Period.Ind
      Sex      <- input$Sex.Ind

      Data.fig     <- Results.LYL[Results.LYL$Period == Period1 &
                                  Results.LYL$Sex == Sex,]

      Data.fig$LYL <- round(Data.fig$LYL,3)

      p <- ggplot(Data.fig, aes(Age, label=LYL))+
        ggtitle(paste('Life years lost for the period', Period1))+
        geom_ribbon(aes(ymin = ymin,ymax =ymax, group=(Cause), fill=Cause),
                    position = 'identity')+
        theme_light()+
        theme(text = element_text(size=10),
              axis.text.x = element_text(angle=45, hjust=1))+
        labs(x = "Age", y = "Probability of surviving and LYL",size=10)

      ggplotly(p,tooltip = c('Cause','Age','LYL'))

    })
```

Finally, we construct our table summarizing the results. We use the `renderDataTable` function to display the results.

```
output$LYL.table = renderDataTable({

      Period1  <- input$Period.Ind
      Sex      <- input$Sex.Ind

      Data.table <- Table.LYL[Table.LYL$Period == Period1 &
                                Table.LYL$Sex == Sex,]

      datatable(Data.table[,c(1,3)], options = list(paging=FALSE,
                                                    ordering=T,
                                                    dom = 't'),
                rownames = F,caption = 'Total life years lost by cause')
    })
```