

DATA SOCIETY®

Intro to Rshiny - Part 3

*"One should look for what is and not what he thinks should be."
-Albert Einstein.*

Warm up

Before we start, check out this blog on dashboards for R Shiny apps: <https://www.r-bloggers.com/2021/02/powerful-dashboard-frameworks-for-r-shiny-apps-in-2021/>

Welcome back

- In the previous classes we built simple Shiny apps with **static content** using different input and output widgets
- Today we will build **interactive Shiny apps**, and review what we've learned so far during the course

Recap

- A Shiny app usually contains two parts:
 - **UI:** controls the layout and appearance of the app
 - **Server:** contains the logic needed to build the app
- When building an application there are four elements that you can customize:
 - **Outputs:** create different output elements, such as plots or tables
 - **Inputs:** use various input widgets the user can utilize in the application
 - **Reactivity:** define how outputs get updated based on the inputs the user chooses
 - **Layout:** decide what your app should look like; use a default one or create your own

Module completion checklist

Objective	Complete
Configure reactive output	
Configure the app to isolate part of the app from regenerating	
Implement different layouts to organize the shiny widgets	

Customizing your application

- When building an application there are four elements that you can customize:
 - **Outputs:** create different output elements, such as plots or tables
 - **Inputs:** use various input widgets the user can utilize in the application
 - **Reactivity:** define how outputs get updated based on the inputs the user chooses
 - **Layout:** decide what your app should look like; use a default one or create your own

What is reactivity?

- Reactivity makes the RShiny apps **responsive**
- It lets the app **update itself** whenever the user makes a change in the input
- With reactivity, we can:
 - create more **sophisticated and efficient Shiny apps**
 - avoid errors that come from misusing reactive values
- For every input the user changes, we get an updated output

Connecting the UI to the server

- Open the files you were working from last session
- We'll start by connecting the checkbox input to the plot output in our server by adding this line to our server script before the plot is created

```
# Keep only the regions selected by the user.  
# `input$region` is a list of regions selected by user.  
region_household = subset(region_household,  
                           region %in% input$region)
```


Connecting the UI to the server

- Our server script will look like this
- Our UI script remains the same

```
library(shiny)
library(dplyr)
library(ggplot2)

# Load the Costa Rican data.
load("region_household.Rdata")
# Define server logic.
server <- function(input, output) {

output$densityplot<-
  renderPlot({ #<- function to create plot object to send to UI
    # Keep only the regions selected by the user.
    region_household = subset(region_household,
                              region %in% input$region)

    # Create density plot.
    ggplot(region_household, #<- set data
            aes(x = total_in_household, #<- map `x` value`
                fill = region )) + #<- map fill
      geom_density(alpha = 0.3) + #<- adjust density fill
      labs(title = "Density of number of people in a household by region") +
      facet_wrap (~ region, #<- make facets by 'region'
                 ncol = 3) #<- set a 3-column grid

  }) # end of renderPlot
}# end of server
```

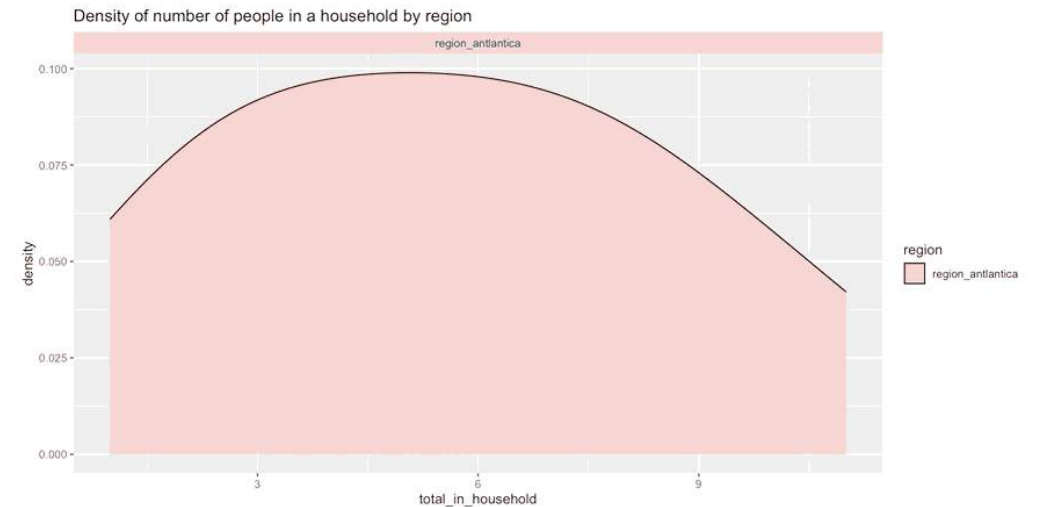
Connecting the UI to the server

- Navigate to the `introduction-to-Rshiny-code/10-interactive-plot` folder
- We now have an interactive plot which changes with user input
- Notice that the plot changes as soon as the user inputs changes, which is not always ideal
- We will try to fix this next

Costa Rican Data

Select Region

- ☒ Atlantica
- ☐ Brunca
- ☐ Central
- ☐ Chorotega
- ☐ Huetar
- ☐ Pacifico



Module completion checklist

Objective	Complete
Configure reactive output	✓
Configure the app to isolate part of the app from regenerating	
Implement different layouts to organize the shiny widgets	

Configure the app to isolate part of the app

- Reactive apps regenerate and run the entire code every time the user input is changed
- **What if we don't want the entire app to regenerate every time?**
- We would want to **isolate** the part of the app from regenerating
- We can use the `isolate()` function to avoid re-rendering the app every time

Observing an event

- **What if we want some action to trigger a new action?**
- We want to observe an action from the user side and then trigger a response from the server side
- We can use `observeEvent()` function for such cases
- In our density plot example, we will:
 - **isolate** the part of the app that we do not want re-rendering
 - create an `actionButton` to trigger the rendering of plot event

Adding the action button to UI

```
library(shiny)

# Define UI for application.
ui<- fluidPage(      #<- fluid pages scale their components in realtime to fill all available browser
width
  titlePanel("Costa Rican Data"), #<- application title
  checkboxGroupInput("region", label = h3("Select Region"),
                    choices = list("Atlantica" = "region_atlantica",
                                   "Brunca" = "region_brunca",
                                   "Central" = "region_central",
                                   "Chorotega" = "region_chorotega",
                                   "Huetar" = "region_huetar",
                                   "Pacifico" = "region_pacifico"
                                   ),
                    selected = "region_atlantica"),

  # Action button to trigger the event.
  actionButton(inputId = "submit", #<- input ID
               label = "Submit"), #<- input label

  plotOutput("densityplot") #<- `scatterplot` from server converted to output element
) #<- end of fluidPage
```

Adding observeEvent() and isolate() to server

```
# Load the Costa Rican data.
load("region_household.Rdata")

# Define server logic.
server <- function(input, output) {

  observeEvent(input$submit,                #<- observe monitors the submit button's value

  output$densityplot<-
    renderPlot({                             #<- function to create plot object to send to UI
      isolate({                               #<- function to isolate this part of app from regenerating
        # Keep only the regions selected by the user.
        region_household = subset(region_household,
                                   region %in% input$region) #<- `input$region` is a list of regions selected
        by user
        # Create density plot.
        ggplot(region_household,              #<- set data
                 aes(x = total_in_household,  #<- map `x` value`
                     fill = region )) +       #<- map fill
          geom_density(alpha = 0.3) +         #<- adjust density fill
          labs(title = "Density of number of people in a household by region") +
          facet_wrap (~ region,               #<- make facets by 'region'
                      ncol = 3)              #<- set a 3-column grid
      }) # end of isolate
    }) # end of renderPlot
  } # end of observeEvent
} # end of server
```

Configure the app to isolate part of the app

- Navigate to the `introduction-to-Rshiny-code/11-interactive-action-button` folder
- We see that the plot is rendered only when the submit button is clicked

Costa Rican Data

Select Region

- ☒ Atlántica
- ☐ Brunca
- ☐ Central
- ☐ Chorotega
- ☐ Huetar
- ☐ Pacífico

Submit

Knowledge check 1



Exercise 1



Module completion checklist

Objective	Complete
Configure reactive output	✓
Configure the app to isolate part of the app from regenerating	✓
Implement different layouts to organize the shiny widgets	

Customizing your application

- When building an application, there are four elements that you can customize:
 - **Outputs:** create different output elements, such as plots or tables
 - **Inputs:** use various input widgets the user can utilize in the application
 - **Reactivity:** define how outputs get updated based on the inputs the user chooses
 - **Layout:** decide what your app should look like; use a default one or create your own

Panels and layouts in Shiny

- **Panels and layouts** are used for arranging our input and output widgets in the UI
- Panels include:

```
conditionalPanel()  
fixedPanel()  
inputPanel()  
mainPanel()  
navlistPanel()  
sidebarPanel()  
titlePanel()  
tabPanel()  
tabsetPanel()
```

- We have used `titlePanel()` to display the title of the app
- We will create an app with `tabPanel()` and `tabsetPanel()` today
- Feel free to understand the functionality of other panels with the helper function


```
?panel_name
```

Layouts in Shiny

- The layouts combine multiple elements into a single element that contains its own panel functions
- The 5 most popular layouts include:
 - `fluidRow()` - arrange the widgets in a row
 - `flowLayout()` - lay out elements in a left-to-right, top-to-bottom arrangement
 - `sidebarLayout()` - arrange the widget with a sidebar and a main bar
 - `splitLayout()` - lay out elements horizontally, dividing the available horizontal space into equal parts (by default)
 - `verticalLayout()` - create a container that includes one or more rows of content

Layouts in Shiny

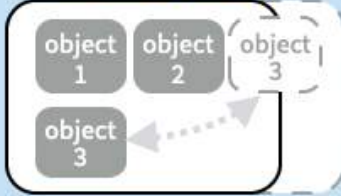
fluidRow()



The diagram shows a container with three columns. The top row contains three boxes labeled 'column', 'row', and 'col'. The bottom row contains a single box labeled 'column'.

```
ui <- fluidPage(  
  fluidRow(column(width = 4),  
    column(width = 2, offset = 3)),  
  fluidRow(column(width = 12))  
)
```


flowLayout()



The diagram shows three boxes labeled 'object 1', 'object 2', and 'object 3' arranged in a flow layout. 'object 1' and 'object 2' are in the top row, and 'object 3' is in the bottom row. A dashed arrow points from 'object 2' to 'object 3'.

```
ui <- fluidPage(  
  flowLayout( # object 1,  
    # object 2,  
    # object 3  
  )  
)
```


sidebarLayout()



The diagram shows a container with two panels. The left panel is labeled 'side panel' and the right panel is labeled 'main panel'.

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(),  
    mainPanel()  
  )  
)
```

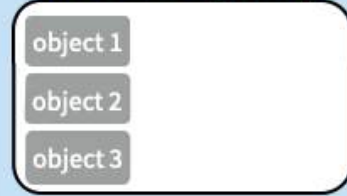
splitLayout()



The diagram shows a container with two boxes labeled 'object 1' and 'object 2' side-by-side.

```
ui <- fluidPage(  
  splitLayout( # object 1,  
    # object 2  
  )  
)
```

verticalLayout()



The diagram shows a container with three boxes labeled 'object 1', 'object 2', and 'object 3' stacked vertically.

```
ui <- fluidPage(  
  verticalLayout( # object 1,  
    # object 2,  
    # object 3  
  )  
)
```

Adding sidebarLayout() to our app: UI

```
library(shiny)

# Define UI for application.
ui<- fluidPage(
  #<- fluid pages scale their components in realtime to fill all available browser width
  titlePanel("Costa Rican Data"), #<- application title
  sidebarLayout(
    #<- layout function
    sidebarPanel(
      checkboxGroupInput("region", label = h3("Select Region"),
        choices = list("Atlantica" = "region_antlantica",
          "Brunca" = "region_brunca",
          "Central" = "region_central",
          "Chorotega" = "region_chorotega",
          "Huetar" = "region_huetar",
          "Pacifico" = "region_pacifico"
        ),
        selected = "region_antlantica"),
      actionButton(inputId = "submit", #<- input ID
        label = "Submit") #<- input label
    ), #<- end of sidebarPanel
  mainPanel(
    plotOutput("densityplot") #<- `densityplot` from server converted to output element
  ) #<- end of mainPanel
) #<- end of sidebarLayout
) #<- end of fluidPage
```


Adding sidebarLayout() to our app: server

- Ideally, since we are only changing the layout of the UI we should not have to change the server script at all
- However, since the sidebar panel will decrease the width of our plot area we will make our plot facet grid have 2 columns instead of 3

```
# Create density plot.
ggplot(region_household,           #<- set data
  aes(x = total_in_household,      #<- map `x` value`
      fill = region )) +          #<- map fill
  geom_density(alpha = 0.3) +      #<- adjust density fill
  labs(title = "Density of number of people in a household by region") +
  facet_wrap (~ region,            #<- make facets by 'region'
             ncol = 2)            #<- set a 2-column grid
```

Our final app

- Navigate to the `introduction-to-Rshiny-code/12-sidebar` folder
- This is how our final application looks!

RShiny: final takeaways

- RShiny is great for those with limited app design experience
 - *Let the built-in layouts do the work for you*
- It is also great for those with app design experience
 - *Allows for customization down to the smallest detail*
- Be careful of the interaction between your inputs and outputs
 - *Isolate and control the parts you do not want regenerating each time*
- RShiny can be integrated with most open-source packages like `ggplot`, `highcharter`, and `visNetwork`
 - *Integrate the packages to create well-designed and customizable apps!*

Knowledge check 2



Exercise 2



Module completion checklist

Objective	Complete
Configure reactive output	✓
Configure the app to isolate part of the app from regenerating	✓
Implement different layouts to organize the shiny widgets	✓

Review

- We've covered the following topics in this course. Any questions?

Topic	Complete
The exploratory data analysis (EDA) cycle and the differences between static and interactive visualizations	✓
Building static univariate/bivariate/multivariate plots	✓
Building plots with ggplot2	✓
Preparing and transforming data with the tidyverse package	✓
Creating interactive visualizations, network visualizations, and motion maps	✓
Building an RShiny application	✓

What next?

- Now you can put your new skills to work! You can select a dataset and use it to:
 - Build basic static plots with the columns of your choice
 - Transform data with `tidyverse`
 - Build complex and interactive plots with `ggplot2` and `highcharter`
 - Create an Rshiny application to present your visualizations and findings

General approach to creating visualizations

- We have listed down the general steps you can apply while working with data to create various types of visualizations:
- **Load and clean the data**
 - set your directories
 - load the selected dataset
 - clean or subset the data, as necessary
- **Basic visualizations**
 - a univariate plot to view the distribution of a single variable
 - a bivariate plot to check the relationship between two variables
 - a multivariate plot for three or more variables

General approach to creating visualizations (cont'd)

tidyverse, ggplot2, and highcharter

- Transforming your data with the `tidyverse` package to get an aggregate table
- Using `ggplot2` to build compound visualizations and customize the theme, label, and title
- Using `highcharter` to build interactive plots

Rshiny application

- Create a Rshiny application with UI and Server
- Start customizing the app using one or more of the following:
 - any type of an input widget
 - isolate a part from regenerating
 - visualizations of the dataset that you want to present
 - a short paragraph with the findings you get from the visualizations

Selecting a dataset

- We've provided two datasets for you to practice with:
 - TB dataset (TB_data.csv) *additional information*
 - Diabetes (diabetes.csv) *additional information*
- Take a few minutes to review the datasets and additional information provided to determine which dataset you'd like to work with
- Alternatively, some of you may have other datasets in mind
 - Take your time to make sure you are familiar with the data they contain

Next steps after EDA

- After performing EDA and finding patterns with visualizations, the next logical step is data modeling
- Supervised machine learning is the task of predicting or classifying an output based on predictors, or input variables
- Below are some examples of how you could apply these algorithms in business scenarios

Question to answer	Real world example
What is the value based on predictors?	Predicting the number of bikes rented based on the weather
What category is this in?	Anticipating if your customer is pregnant, remodeling, just got married, etc.
What is the probability that something is in a given category?	Determining the probability that a piece of equipment will fail, or that someone will buy your product

This completes our module
Congratulations!