DATA SOCIETY®

Intro to Rshiny - Part 2

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

Welcome back!

- In the last module, we introduced Shiny apps and we'll continue on that topic today.
- Let's start by taking a minute to browse the ***Shiny Gallery*** and find an interesting app to explore.

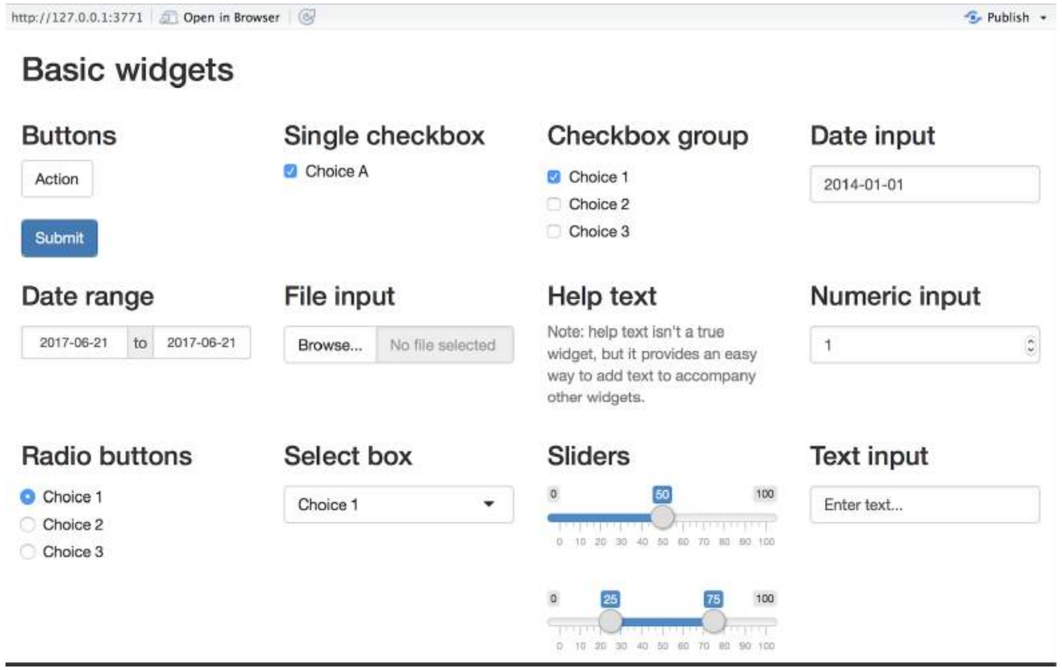
Module completion checklist

Objective	Complete
Create and integrate action buttons, sliders into Rshiny	
Configure and integrate single checkbox, groups into Rshiny	
Configure numeric input box in Rshiny	

DATA SOCIETY Ⓒ 2021

Recap: built-in widgets

- Shiny has built-in **widgets** for user input
- The ***widget gallery*** is a useful resource to see what widgets are available and get the code for each widget



Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

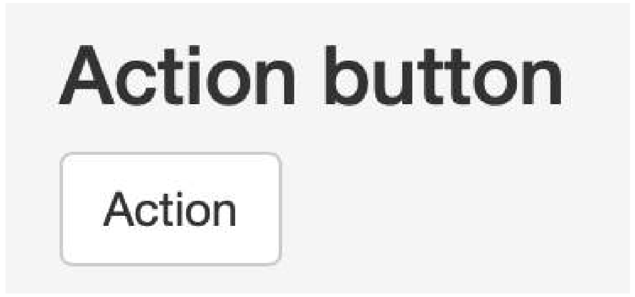
Inputs: widgets we'll explore

Output	Functions used
Action button	actionButton
Slider	sliderInput
Slider range	sliderInput
Single checkbox	checkboxInput
Checkbox group	checkboxGroupInput
Numeric input	numericInput
Text input	textInput
Radio buttons	radioButtons

- We'll explore how to create each of these in detail today!

Inputs: action button

- **What it looks like**



- **What it does**
  - Functions like the 'Enter' key on your key board
- **When it is used**
  - Whenever you want the user to confirm an action, such as update a graph or perform a calculation

Creating action buttons in R

- Here's how we will create an action button widget. It has two arguments:
  - **Input id:** The id of the action button
  - **Label:** The text or label to be displayed on the action button

```
# This function will not generate an action button on its own.
# It needs to be added to the base UI script we created earlier.

actionButton("change_in_action_id", "Click here!")
```

- There's an another alternative to action buttons, called an **action link**
- It has the format of a hyperlink, but behaves the same way as an action button and has the same input arguments

```
# This function will not generate an action link on its own.
# It needs to be added to the base UI script we created earlier.

actionLink("change_in_action_id", "Click here!")
```

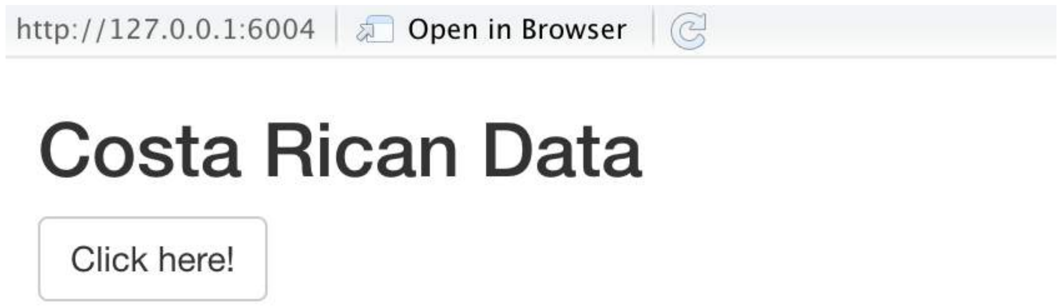
Adding an action button to our base app: UI

- We will add `actionButton()` to our UI script for now

```
library(shiny)

# Define UI for application.
ui<- fluidPage(        #<- fluid pages scale their components in real time to fill all available
browser width
   titlePanel("Costa Rican Data"), #<- application title
   actionButton("button", "Click here!") #<- add action button

  ) #<- end of fluidPage
```

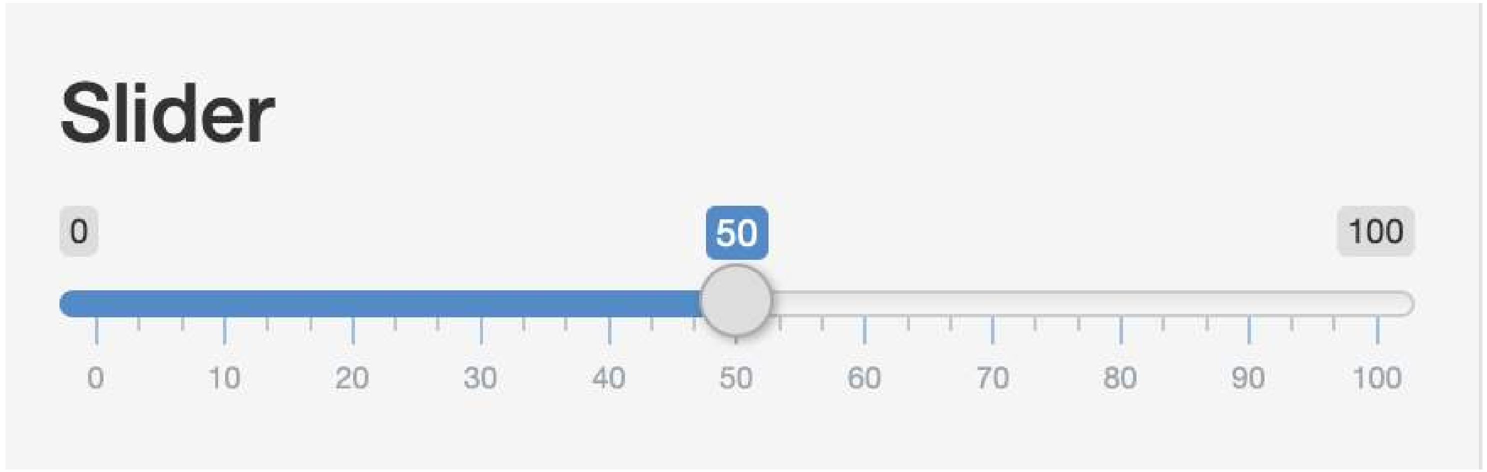
Adding an action button to our base app - cont'd

- Keeping the server script the same, run the app with the action button in it
- Navigate to `introduction-to-Rshiny-code/4-action-button` folder
- At this point, no action is triggered when we click the action button since there's no reactivity associated with it
- It can be configured based on the input id associated with the action button/link
- We'll learn more about reactivity in general in the next session!



Inputs: slider

- **What it looks like**



- **What it does**
  - Lets the user select a specific number by moving the slider with the mouse
- **When it is used**
- Whenever you want the user to select a number, such as:
  - Select number of bins in a histogram
  - Select number of rows to be displayed in a table
  - Select a particular year's data to be displayed

Creating a slider in R

- Here's how we will create a slider widget

```r
# This function will not generate a slider on its own.
# It needs to be added to the base UI script we created earlier.

# slider examples
sliderInput("Example-1", "Basic Integer slider", #<- add Input Id and label
min = 0, max = 500,  #<- specify max and min values
value = 250), #<- default value to display when we run the app

# slider with step
sliderInput("Example-2", " Slider with step",
min = 0, max = 2,
value = 1, step = 0.5), #<- set step as 0.5

# slider with range specification
sliderInput("Example-3", "Slider with range",
min = 1, max = 500,
value = c(100,250)),  #<- specify range to be displayed when we run the app

# Slider with custom currency formatting and animation
sliderInput("Example-4", "Custom slider with animation",
min = 0, max = 1000,
value = 0, step = 250,
pre = "$", sep = ",",  #<- specify pre-fix and separator to display in the slider
animate = TRUE)  #<- configure animate button
```

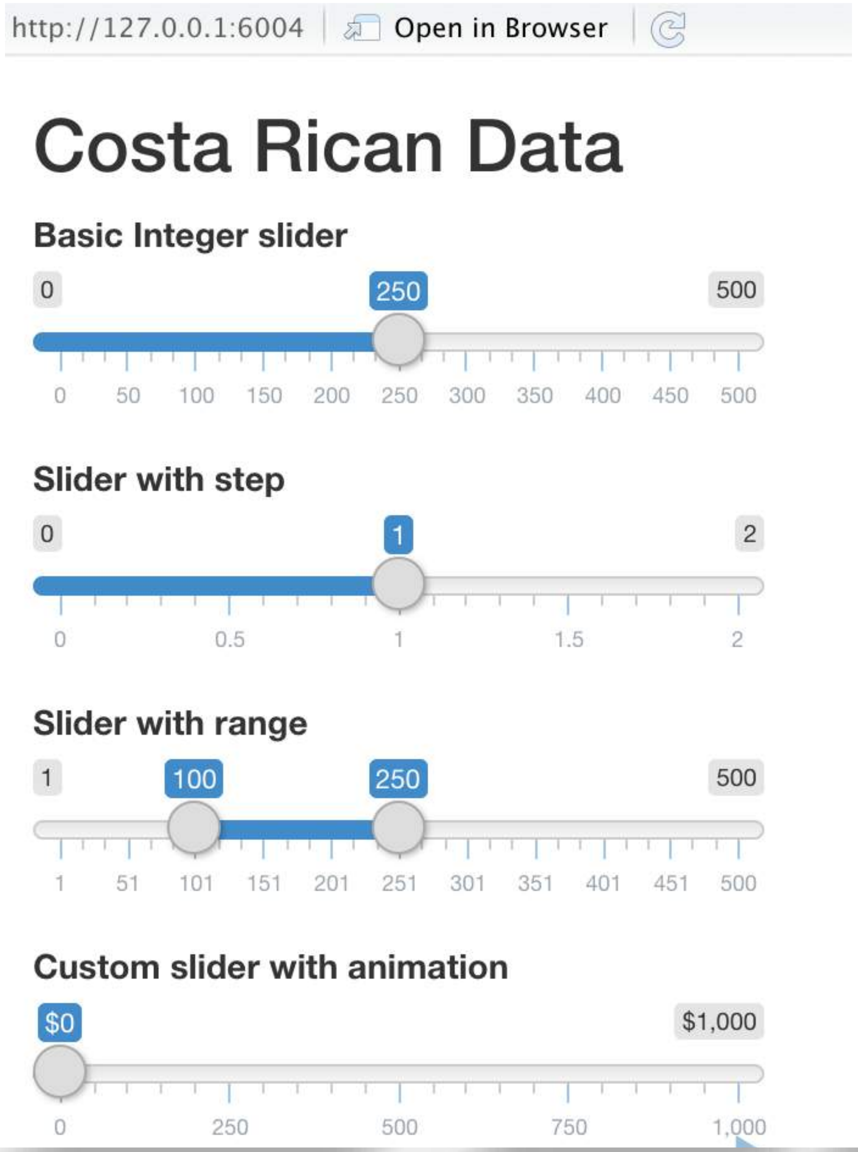
Adding a slider to our base app: UI

- We will add `sliderInput` to our UI script for now

```
library(shiny)

# Define UI for application.
ui<- fluidPage(         #<- fluid pages scale their components in real time to fill all available
browser width
  titlePanel("Costa Rican Data"), #<- application title
  # slider examples
  sliderInput("Example-1", "Basic Integer slider", #<- add Input Id and label
              min = 0, max = 500,  #<- specify max and min values
              value = 250), #<- default value to display when we run the app

  # slider with step
  sliderInput("Example-2", " Slider with step",
              min = 0, max = 2,
              value = 1, step = 0.5), #<- set step as 0.5

  # slider with range specification
  sliderInput("Example-3", "Slider with range",
              min = 1, max = 500,
              value = c(100,250)),  #<- specify range to be displayed when we run the app

  # Slider with custom currency formatting and animation
  sliderInput("Example-4", "Custom slider with animation",
              min = 0, max = 1000,
              value = 0, step = 250,
              pre = "$", sep = ",",  #<- specify pre-fix and separator to display in the slider
              animate = TRUE)  #<- configure animate button
) #<- end of fluidPage
```

Adding a slider to our base app - cont'd

- Keeping the server script the same, run the app with the slider in it
- Navigate to `introduction-to-Rshiny-code/5-slider` folder



Knowledge check 1

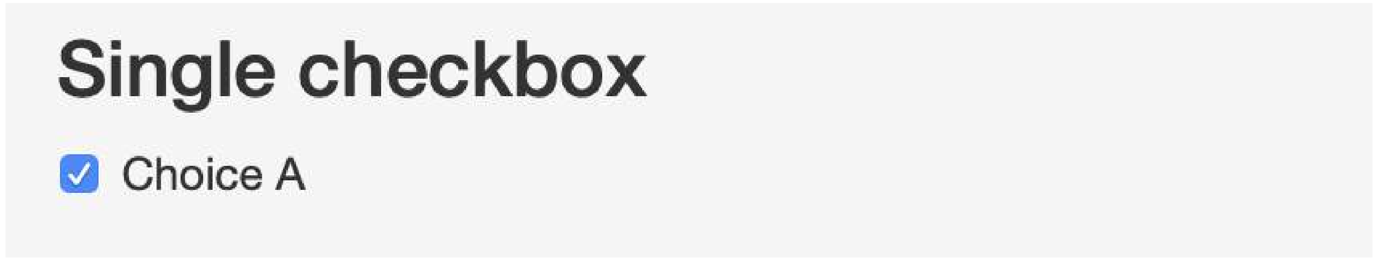
Exercise 1

Module completion checklist

Objective	Complete
Create and integrate action buttons, sliders into Rshiny	✔
Configure and integrate single checkbox, groups into Rshiny	
Configure numeric input box in Rshiny	

Inputs: single checkbox

- **What it looks like**



- **What it does**
  - Lets the user select/unselect an option
- **When it is used**
- User should be able to toggle an option "on" and "off", such as:
  - Select if individual observations should be shown in graph
  - Select if table should show a header
  - Select if data should be updated automatically

Creating checkbox in R

- Here's how we will create a simple checkbox input widget. It has four arguments:
  - **Input id:** The id of the checkbox
  - **Label:** The text or label to be displayed beside the checkbox
  - **Value:** Initial value (`True` OR `False`)
  - **Width:** Width of the input widget (optional)

```r
# This function will not generate a checkbox on its own.
# It needs to be added to the base UI script we created earlier.

ui <- fluidPage(
  checkboxInput("checkbox-input", "Checkbox input", FALSE)
)
```

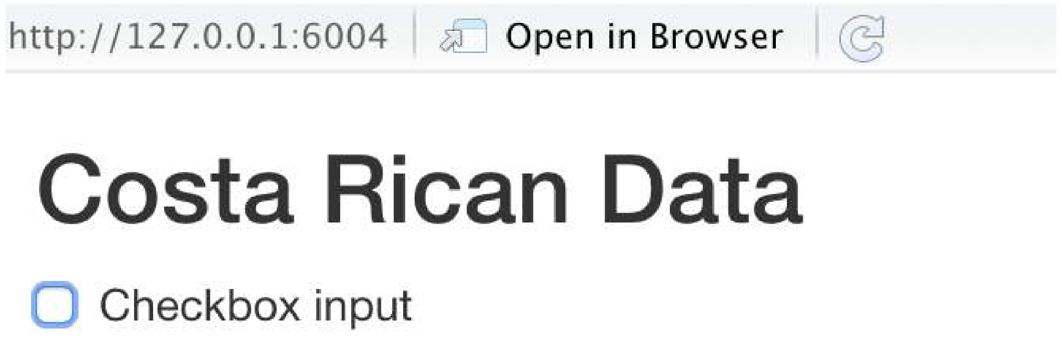
Adding checkbox to our base app: UI

- We will add `checkboxInput()` to our UI script

```
library(shiny)

# Define UI for application.
ui<- fluidPage(        #<- fluid pages scale their components in real time to fill all available
browser width
  titlePanel("Costa Rican Data"), #<- application title
  checkboxInput("checkbox-input", "Checkbox input", FALSE)
) #<- end of fluidPage
```

Adding checkbox to our base app - cont'd

- Keeping the server script the same, run the app with our checkbox input
- Navigate to `introduction-to-Rshiny-code/6-checkbox-widgets` folder



Costa Rican Data

☐ Checkbox input

Data preparation: load the dataset

- Before exploring checkbox groups and radio buttons, we'll quickly create a basic density plot
- We will be using the `region_household` dataset which we used in the last class to create the base app and add some input widgets to it
- This dataset gives us a summary of the total number of households in each Costa Rican region

```
# Set the working directory to the data directory.
setwd(data_dir)

# Load the dataset and view the first few rows.
load("region_household.Rdata")
head(region_household)
```

```
          region total_in_household count_by_region
1 region_central                  1             243
2 region_central                  2             797
3 region_central                  3            1300
4 region_central                  4            1460
5 region_central                  5             931
6 region_central                  6             468
```

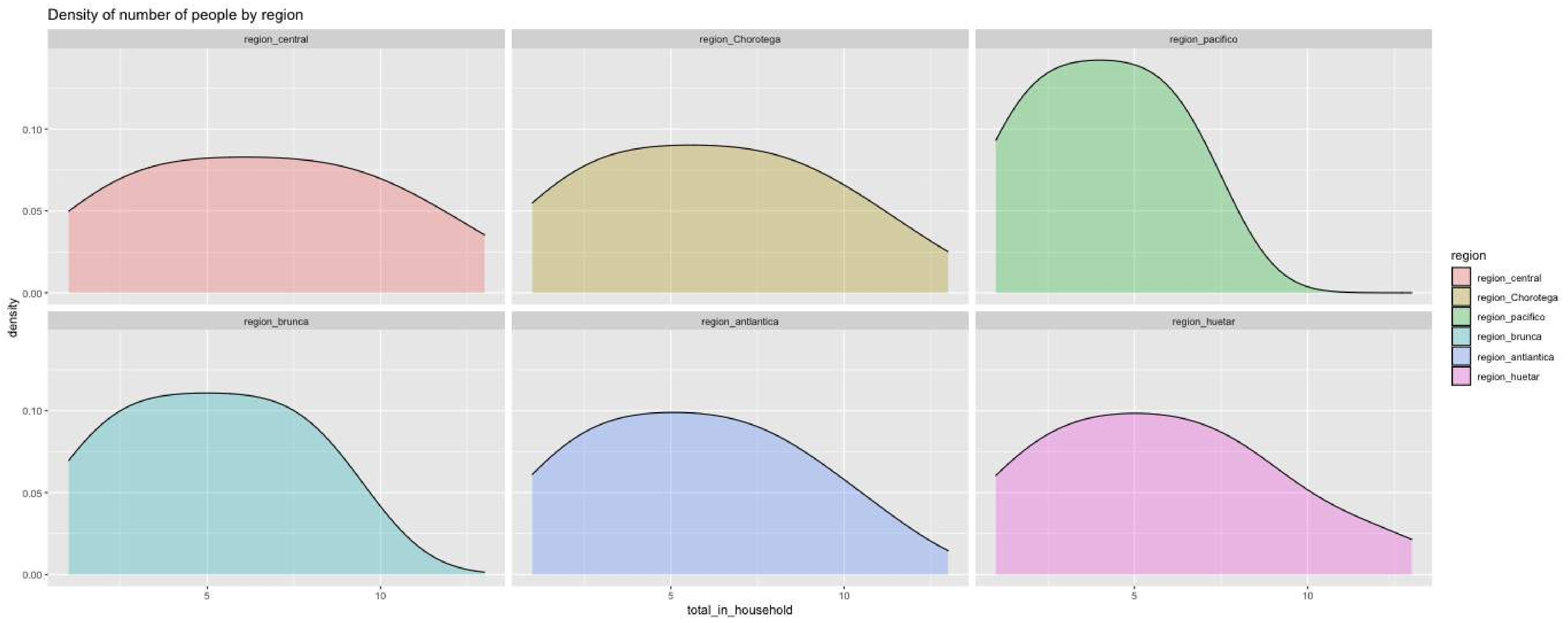
Create static density plot based on regions

- We'll create the density plot to measure `total_in_household` based on region and generate an app using Rshiny
- Then, we'll experiment with checkbox groups and radio buttons on UI and see what they look like on the app!

```r
# This is how our static `ggplot` density plot was created.

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 fill transparency
       labs(title =                   #<- add title
            "Density of number of people by region") +
      facet_wrap (~ region,           #<- make facets by 'region'
                  ncol = 3)           #<- set a 3-column grid
```

Create static density plot based on regions - cont'd

```
density_plot
```



Add density plot to our base app: UI

- We will add the plot object `densityplot` created in the server to our base UI
- We will also update the titles

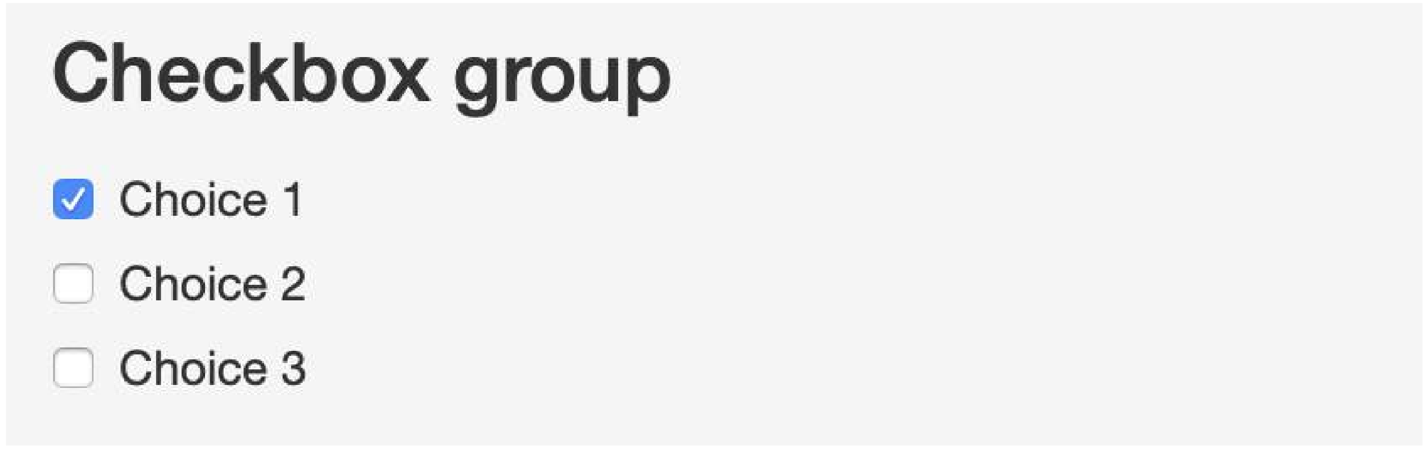
```r
library(shiny)

ui <- fluidPage(

  # Title of the app.
  titlePanel("Costa Rican Data"),

  # Render the output as plot.
  plotOutput(outputId = "densityplot")

)
```

Add density plot to our base app: server

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

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

  # Load the dataset.
  load("region_household.Rdata")

output$densityplot<-
  renderPlot({   #<- function to create plot object to send to UI

  # 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
```

Inputs: checkbox group

- **What it looks like**



Checkbox group

- [x] Choice 1
- [ ] Choice 2
- [ ] Choice 3

- **What it does**
  - Lets the user select more than one option in a group of choices
- **When it is used**
  - To select which age groups to include in graphs
  - To select which countries and gender to include in the analysis

Creating a checkbox group in R

- We will use a checkbox group widget to accept user input for `region`:

```
# This function will not generate a checkbox on its own.
# It needs to be added to the base UI script we created earlier.

checkboxGroupInput(
                   "region",                                        #<- name of the input variable
                   label = h3("Select Region"),                     #<- the title for the widget
                   choices = list("Antlantica" = "region_antlantica", #<- list of choices
                                  "Brunca" = "region_brunca",        #   with choice label
                                  "Central" = "region_central",      #   and choice value
                                  "Chorotega"= "region_Chorotega",
                                  "Huetar" = "region_huetar",
                                  "Pacifico" = "region_pacifico"),
                             selected = NULL)                        #<- initial selected value
```

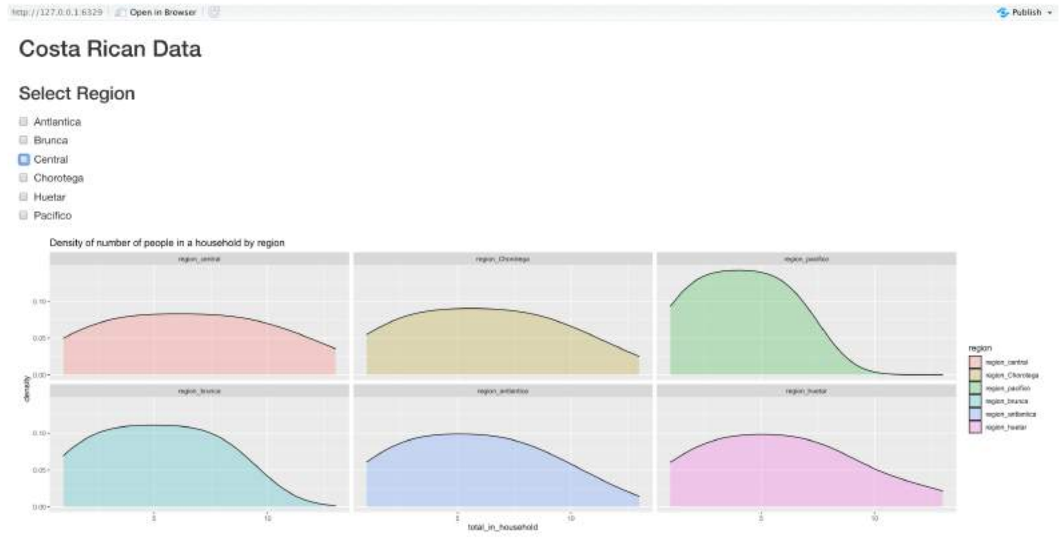
Adding a checkbox group to our base app: UI

- We will add `checkboxGroupInput()` to our UI script

```
library(shiny)

# Define UI for application.
ui<- fluidPage(        #<- fluid pages scale their components in real time to fill all available
browser width
   titlePanel("Costa Rican Data"), #<- application title
   checkboxGroupInput("region", label = h3("Select Region"),
                     choices = list("Antlantica" = "region_antlantica",
                                    "Brunca" = "region_brunca",
                                    "Central" = "region_central",
                                    "Chorotega"= "region_Chorotega",
                                    "Huetar" = "region_huetar",
                                    "Pacifico" = "region_pacifico"
                                    ),
                     selected = "region_antlantica"), #<- set default input
    plotOutput("densityplot")  #<- `scatterplot` from server converted to output element
  ) #<- end of fluidPage
```

Adding a checkbox group to our base app - cont'd

- Keeping the server script the same, run the app with our checkboxGroup input
- Navigate to `introduction-to-Rshiny-code/7-checkbox-group-widgets` folder
- The plot is still static since our input is not linked to the plot. The plots will not change according to the checkboxes selected.



http://127.0.0.1:6329 Open in Browser Publish

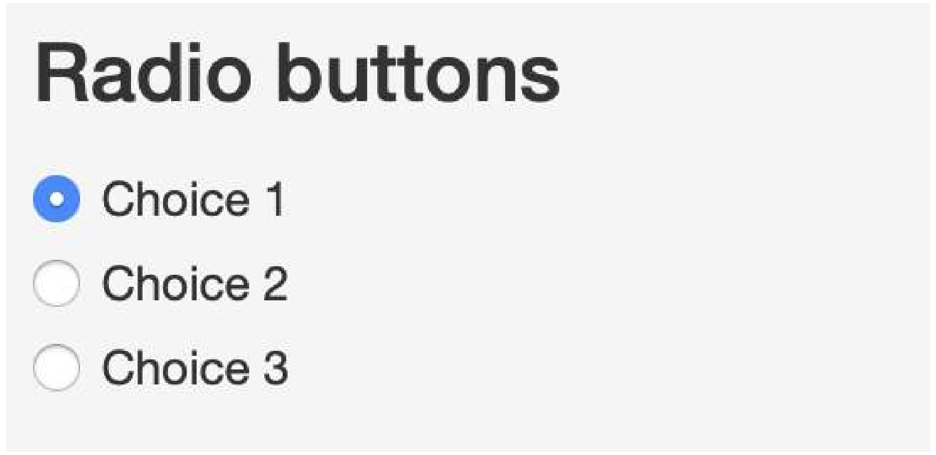
Costa Rican Data

Select Region

- ☐ Antlantica
- ☐ Brunca
- ☐ Central
- ☐ Chorotega
- ☐ Huetar
- ☐ Pacifico

Inputs: radio button

- **What it looks like**



- **What it does**
  - Lets the user select one option out of a group of options
- **When it is used**
  - Let user select the dataset to be displayed in a graph
  - Let user select the type of graph to be displayed
  - Let user select a color in a graph

Creating radio buttons in R

- We will use a radioButtons widget to accept user input for `region`:

```
# This function will not generate a checkbox on its own.
# It needs to be added to the base UI script we created earlier.

radioButtons(
                   "region",                                               #<- name of the input variable
                   label = h3("Select Region"),                            #<- the title for the widget
                   choices = list("Antlantica" = "region_antlantica",     #<- list of choices
                                  "Brunca" = "region_brunca",              #   with choice label
                                  "Central" = "region_central",            #   and choice value
                                  "Chorotega"= "region_Chorotega",
                                  "Huetar" = "region_huetar",
                                  "Pacifico" = "region_pacifico"),
                             selected = NULL)                              #<- initial selected value
```

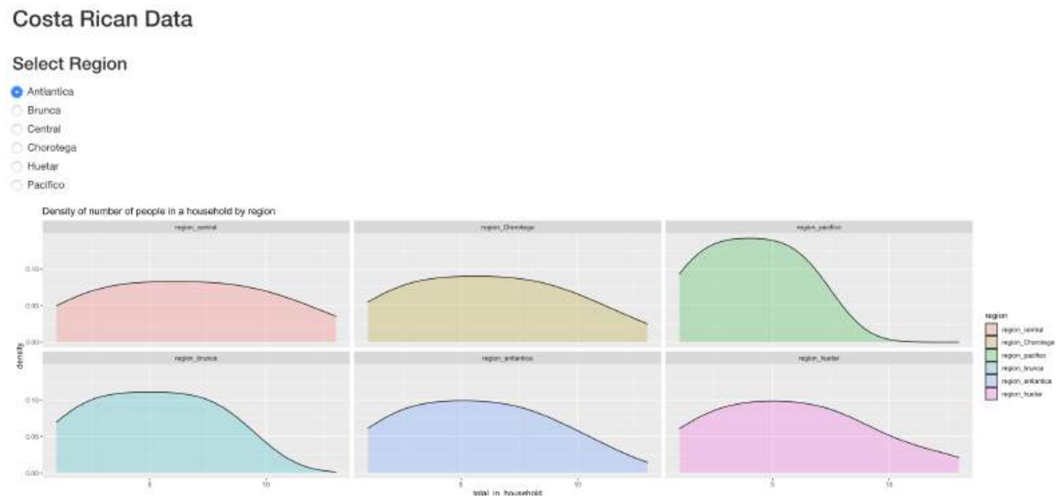
Adding radio buttons to our base app: UI

- We will add `radioButtons()` to our UI script

```
library(shiny)

# Define UI for application.
ui<- fluidPage(        #<- fluid pages scale their components in real time to fill all available
browser width
   titlePanel("Costa Rican Data"), #<- application title
   radioButtons("region", label = h3("Select Region"),
                      choices = list("Antlantica" = "region_antlantica",
                                     "Brunca" = "region_brunca",
                                     "Central" = "region_central",
                                     "Chorotega"= "region_Chorotega",
                                     "Huetar" = "region_huetar",
                                     "Pacifico" = "region_pacifico"
                                     ),
                      selected = "region_antlantica"), #<- set default input
    plotOutput("densityplot")  #<- `scatterplot` from server converted to output element
  ) #<- end of fluidPage
```

Adding radio buttons to our base app - cont'd

- Keeping the server script the same, run the app with our radioButton input
- Navigate to `introduction-to-Rshiny-code/8-radio-button` folder
- The plot is still static since our input is not linked to the plot. The plots will not change according to the button selected.

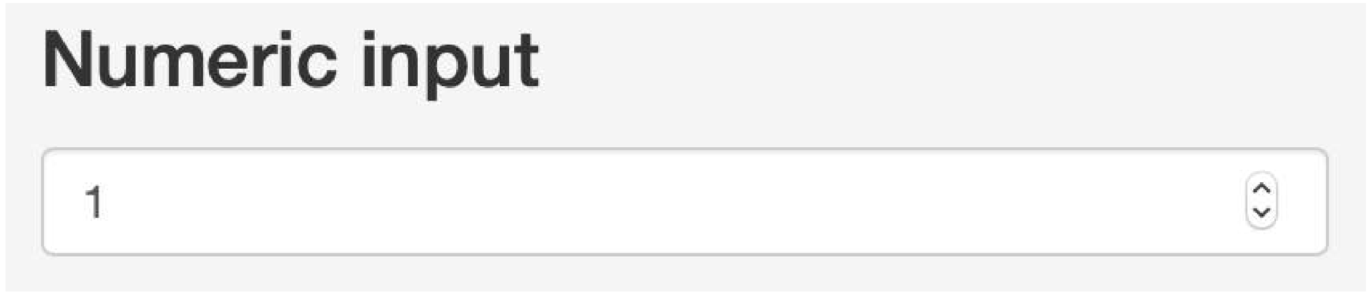


Module completion checklist

Objective	Complete
Create and integrate action buttons, sliders into Rshiny	✔
Configure and integrate single checkbox, groups and radio buttons into Rshiny	✔
Configure numeric input box in Rshiny	

Inputs: numeric input

- **What it looks like**



- **What it does**
  - Lets the user specify a number
- **When it is used**
  - To let users specify year of birth, income level or zip code

Creating a numeric input box in R

- Here's how we will create a numeric input widget

```
# This function will not generate an numeric input on its own.
# It needs to be added to the base UI script we created earlier.

numericInput("contact",  #<- input_id
               "Contact number:", #<- Label to display
               0, #<- default input
               min = 0,  #<- minimum value
               max = 9999999999)  #<- maximum value
```

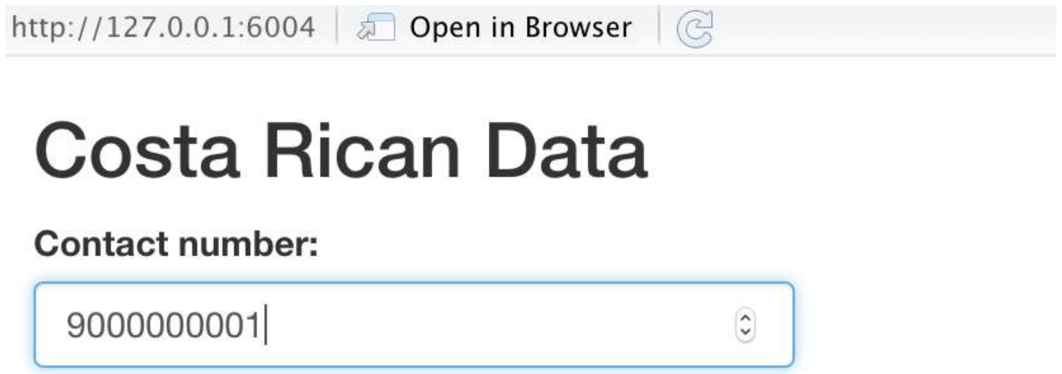
Adding a numeric input box to our base app: UI

- We will add `numericInput()` to our UI script for now

```r
library(shiny)

# Define UI for application.
ui<- fluidPage(        #<- fluid pages scale their components in real time to fill all available
browser width
  titlePanel("Costa Rican Data"), #<- application title
  numericInput("contact",  #<- input_id
               "Contact number:", #<- Label to display
               0, #<- default input
               min = 0,  #<- minimum value
               max = 9999999999)  #<- maximum value
) #<- end of fluidPage
```

Adding a numeric input box to our base app

- Keeping the server script the same, run the app with the action button in it
- Navigate to `introduction-to-Rshiny-code/9-numeric-input` folder



Costa Rican Data

Contact number:

9000000001

Knowledge check 2

Exercise 2

DATA SOCIETY Ⓒ 2021

Module completion checklist

Objective	Complete
Create and integrate action buttons, sliders into Rshiny	✔
Configure and integrate single checkbox, groups and radio buttons into Rshiny	✔
Configure numeric input box in Rshiny	✔

Summary

Today we learned more about RShiny apps:

- action buttons
- sliders
- checkboxes and checkbox groups
- numeric input boxes

In the next module, we will continue looking into RShiny apps. We will cover interactive Shiny apps.

This completes our module

Congratulations!