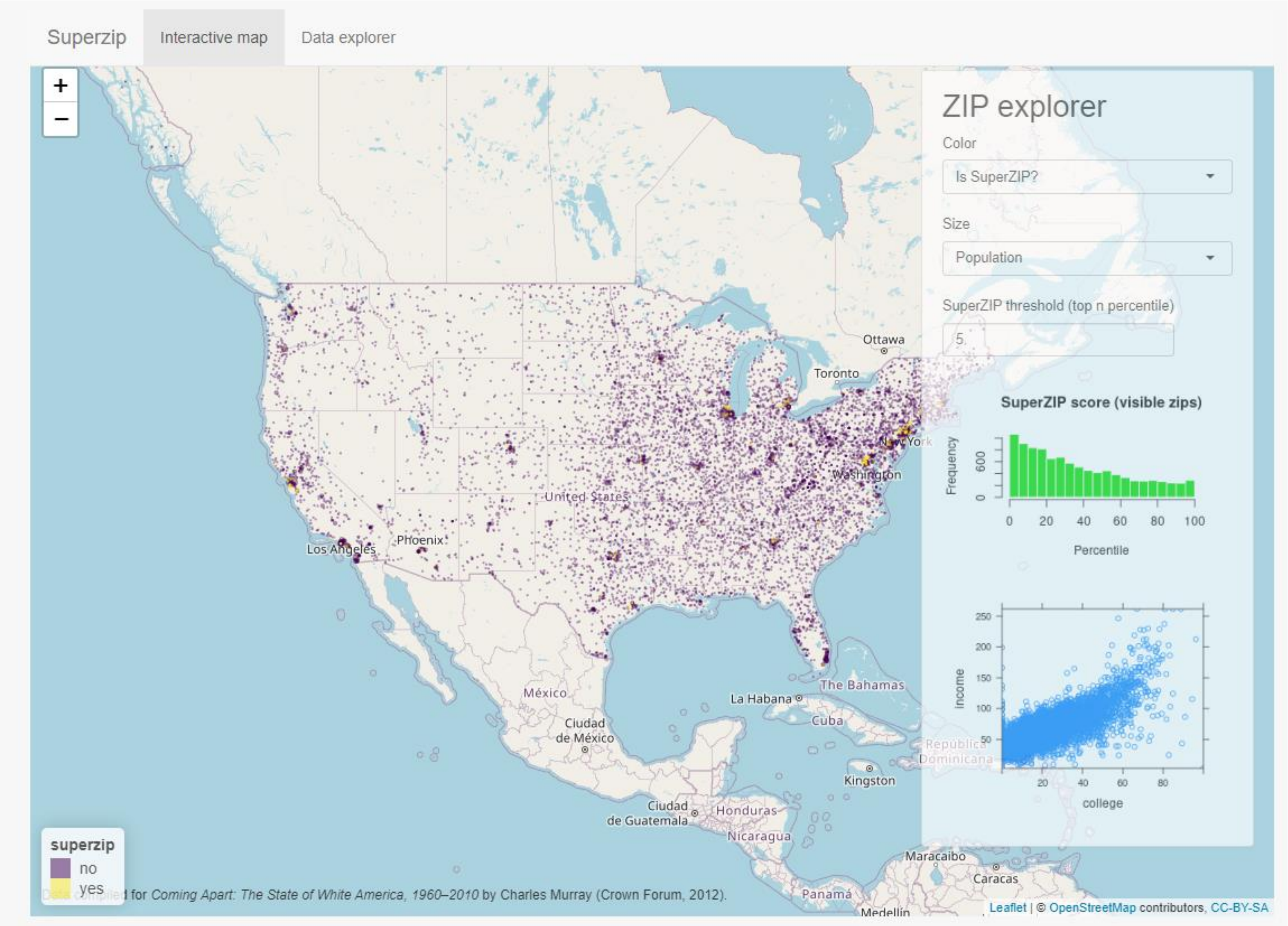
Bigfoot sightings: Learning to create Shiny modules by turning an existing app modular

- Deepsha Menghani

Shiny is a framework for building web applications using R and Python.



Color: Is SuperZIP?

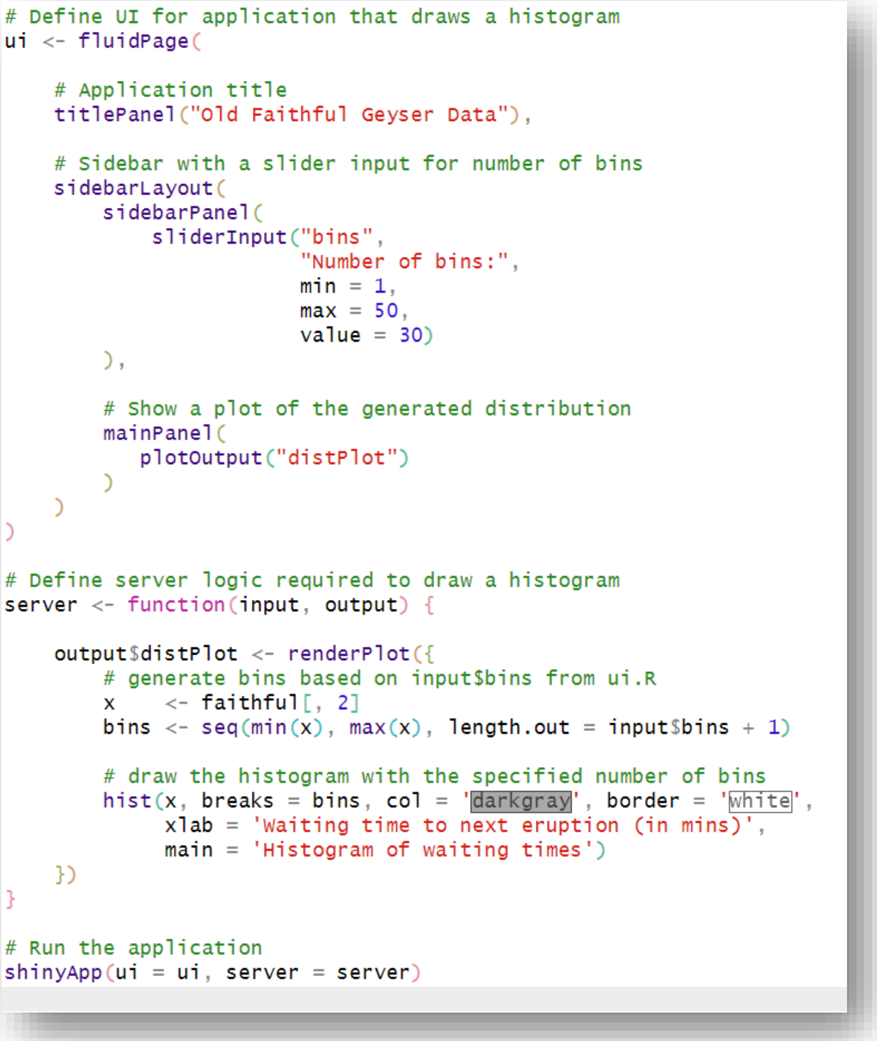
Size: Population

SuperZIP threshold (top n percentile): 5

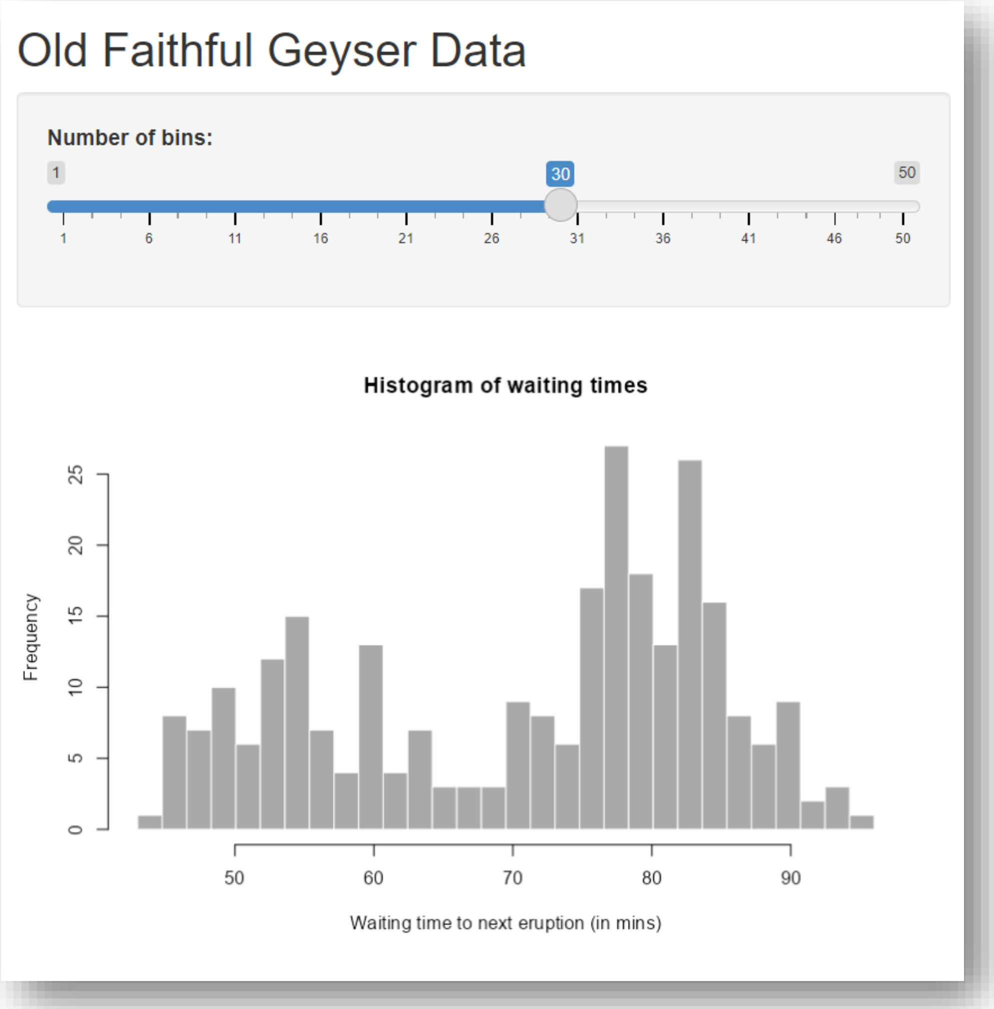
Data compiled for *Coming Apart: The State of White America, 1960–2010* by Charles Murray (Crown Forum, 2012).

Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/

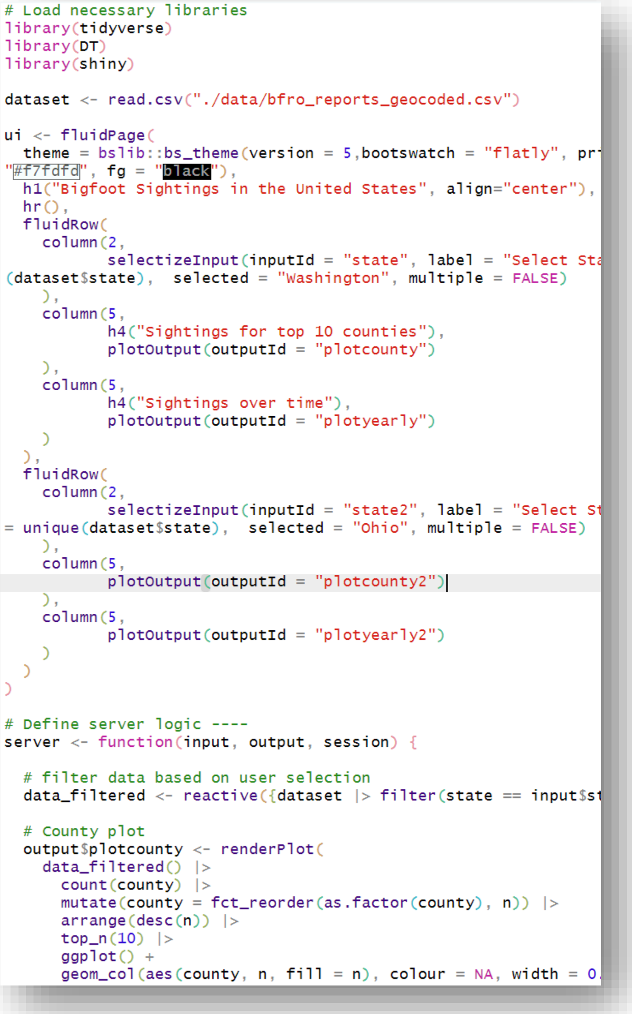
Learning Shiny…. A simple app.R file



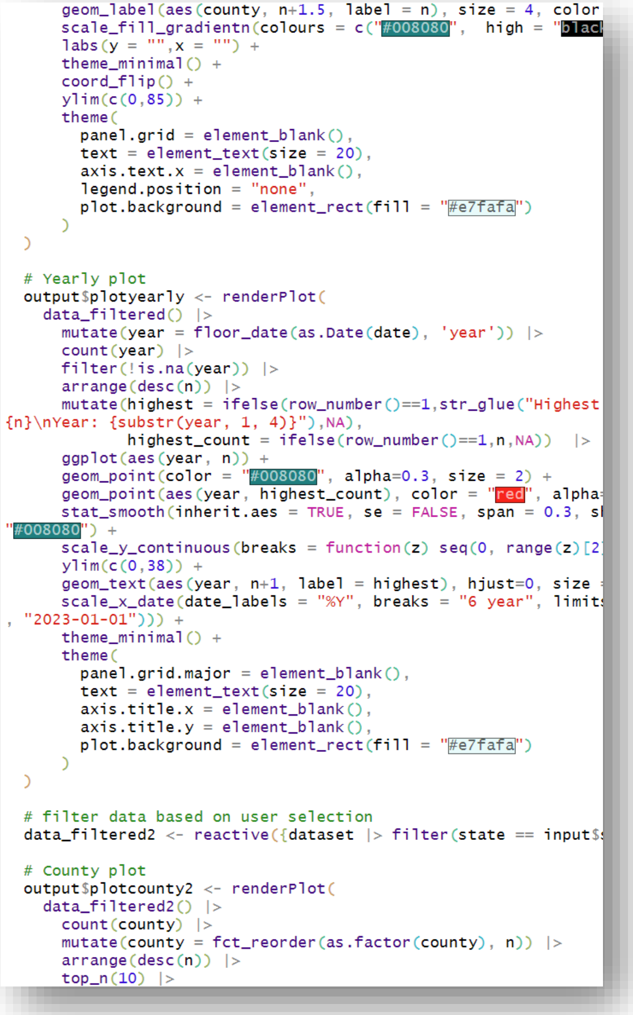
```r
# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Old Faithful Geyser Data"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )
    )
)

# Define server logic required to draw a histogram
server <- function(input, output) {

    output$distPlot <- renderPlot({
        # generate bins based on input$bins from ui.R
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        # draw the histogram with the specified number of bins
        hist(x, breaks = bins, col = 'darkgray', border = 'white',
             xlab = 'Waiting time to next eruption (in mins)',
             main = 'Histogram of waiting times')
    })
}

# Run the application
shinyApp(ui = ui, server = server)
```



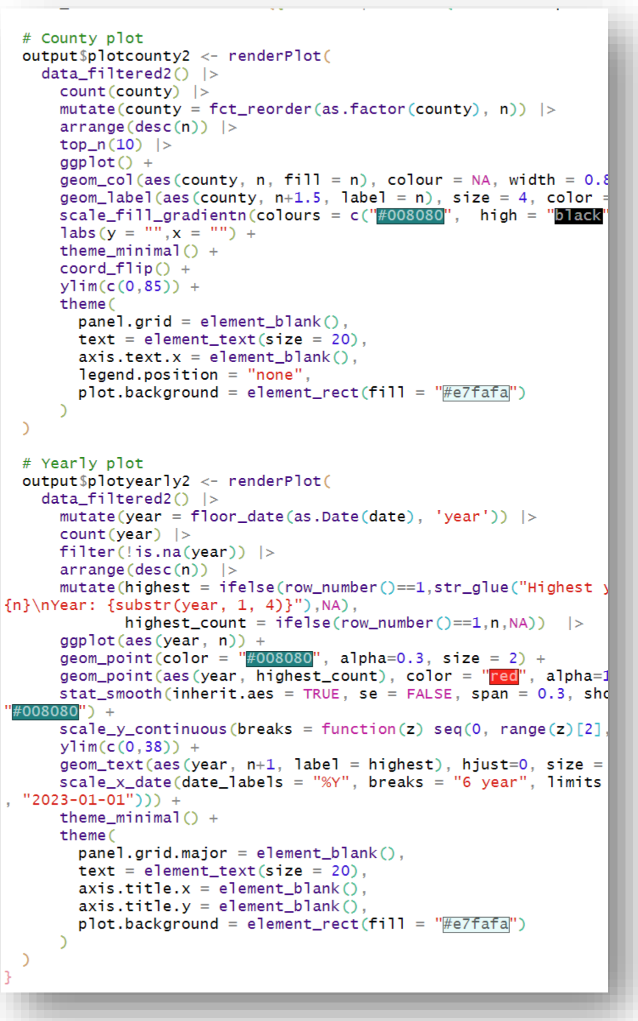
Building Shiny…. A much larger app.R file



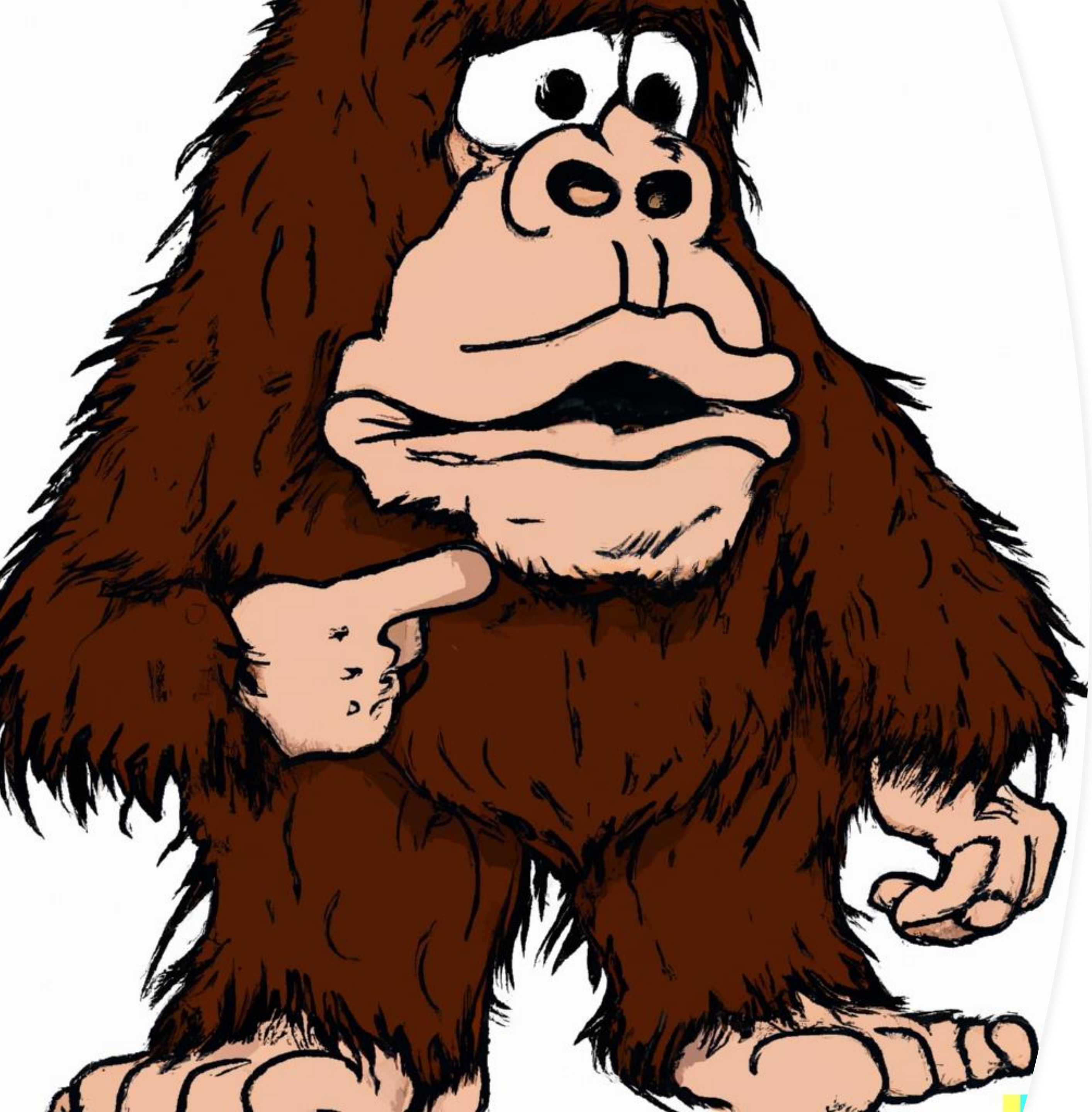
```r
# Load necessary libraries
library(tidyverse)
library(DT)
library(shiny)

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(
  theme = bslib::bs_theme(version = 5,bootswatch = "flatly", pr
"#f7fdfd", fg = "black"),
  h1("Bigfoot Sightings in the United States", align="center"),
  hr(),
  fluidRow(
    column(2,
           selectizeInput(inputId = "state", label = "Select Sta
(dataset$state),  selected = "Washington", multiple = FALSE)
    ),
    column(5,
           h4("Sightings for top 10 counties"),
           plotOutput(outputId = "plotcounty")
    ),
    column(5,
           h4("Sightings over time"),
           plotOutput(outputId = "plotyearly")
    )
  ),
  fluidRow(
    column(2,
           selectizeInput(inputId = "state2", label = "Select St
= unique(dataset$state),  selected = "Ohio", multiple = FALSE)
    ),
    column(5,
           plotOutput(outputId = "plotcounty2")
    ),
    column(5,
           plotOutput(outputId = "plotyearly2")
    )
  )
)

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$st

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      mutate(county = fct_reorder(as.factor(county), n)) |>
      arrange(desc(n)) |>
      top_n(10) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.
```



```r
      geom_label(aes(county, n+1.5, label = n), size = 4, color
      scale_fill_gradientn(colours = c("#008080",  high = "black
      labs(y = "",x = "") +
      theme_minimal() +
      coord_flip() +
      ylim(c(0,85)) +
      theme(
        panel.grid = element_blank(),
        text = element_text(size = 20),
        axis.text.x = element_blank(),
        legend.position = "none",
        plot.background = element_rect(fill = "#e7fafa")
      )
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      mutate(year = floor_date(as.Date(date), 'year')) |>
      count(year) |>
      filter(!is.na(year)) |>
      arrange(desc(n)) |>
      mutate(highest = ifelse(row_number()==1,str_glue("Highest
{n}\nYear: {substr(year, 1, 4)}"),NA),
             highest_count = ifelse(row_number()==1,n,NA))  |>
      ggplot(aes(year, n)) +
      geom_point(color = "#008080", alpha=0.3, size = 2) +
      geom_point(aes(year, highest_count), color = "red", alpha
      stat_smooth(inherit.aes = TRUE, se = FALSE, span = 0.3, sl
"#008080") +
      scale_y_continuous(breaks = function(z) seq(0, range(z)[2
      ylim(c(0,38)) +
      geom_text(aes(year, n+1, label = highest), hjust=0, size
      scale_x_date(date_labels = "%Y", breaks = "6 year", limits
, "2023-01-01"))) +
      theme_minimal() +
      theme(
        panel.grid.major = element_blank(),
        text = element_text(size = 20),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        plot.background = element_rect(fill = "#e7fafa")
      )
  )

  # filter data based on user selection
  data_filtered2 <- reactive({dataset |> filter(state == input$

  # County plot
  output$plotcounty2 <- renderPlot(
    data_filtered2() |>
      count(county) |>
      mutate(county = fct_reorder(as.factor(county), n)) |>
      arrange(desc(n)) |>
      top_n(10) |>
```



```r
  # County plot
  output$plotcounty2 <- renderPlot(
    data_filtered2() |>
      count(county) |>
      mutate(county = fct_reorder(as.factor(county), n)) |>
      arrange(desc(n)) |>
      top_n(10) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8
      geom_label(aes(county, n+1.5, label = n), size = 4, color =
      scale_fill_gradientn(colours = c("#008080",  high = "black"
      labs(y = "",x = "") +
      theme_minimal() +
      coord_flip() +
      ylim(c(0,85)) +
      theme(
        panel.grid = element_blank(),
        text = element_text(size = 20),
        axis.text.x = element_blank(),
        legend.position = "none",
        plot.background = element_rect(fill = "#e7fafa")
      )
  )

  # Yearly plot
  output$plotyearly2 <- renderPlot(
    data_filtered2() |>
      mutate(year = floor_date(as.Date(date), 'year')) |>
      count(year) |>
      filter(!is.na(year)) |>
      arrange(desc(n)) |>
      mutate(highest = ifelse(row_number()==1,str_glue("Highest y
{n}\nYear: {substr(year, 1, 4)}"),NA),
             highest_count = ifelse(row_number()==1,n,NA))  |>
      ggplot(aes(year, n)) +
      geom_point(color = "#008080", alpha=0.3, size = 2) +
      geom_point(aes(year, highest_count), color = "red", alpha=1
      stat_smooth(inherit.aes = TRUE, se = FALSE, span = 0.3, sho
"#008080") +
      scale_y_continuous(breaks = function(z) seq(0, range(z)[2],
      ylim(c(0,38)) +
      geom_text(aes(year, n+1, label = highest), hjust=0, size =
      scale_x_date(date_labels = "%Y", breaks = "6 year", limits
, "2023-01-01"))) +
      theme_minimal() +
      theme(
        panel.grid.major = element_blank(),
        text = element_text(size = 20),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        plot.background = element_rect(fill = "#e7fafa")
      )
  )
}
```



The challenges with a monolith app.R file

- Difficult to keep track of components
- Challenging to debug if something fails
- Harder for someone new to learn and contribute
- Tedious to reuse its elements within itself and other applications

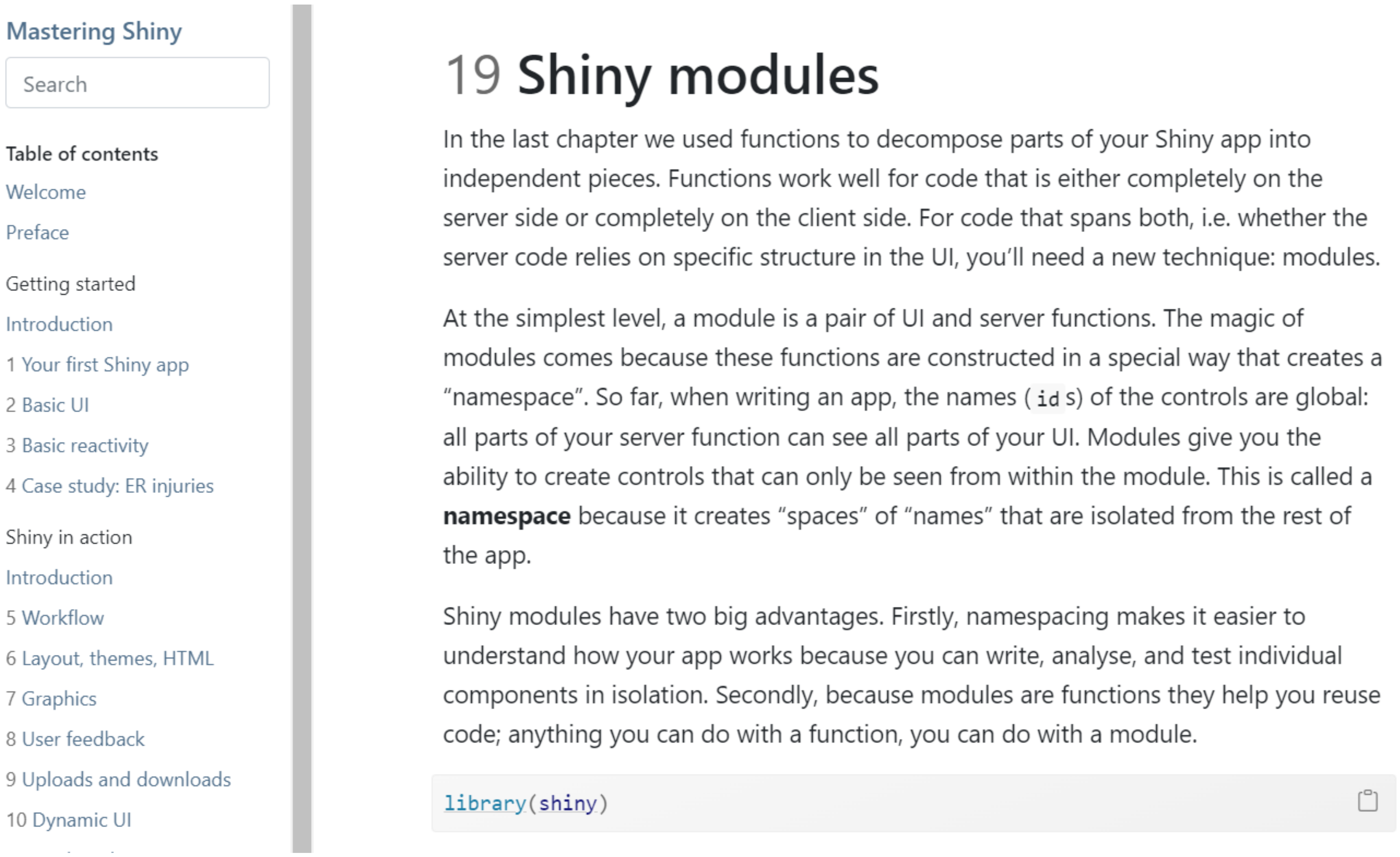
Enter Shiny modules… A reusable piece of Shiny code written as a function

Functions

❤

Shiny

Superpower of Shiny and R functions combined



Mastering Shiny

Search

Table of contents

19 Shiny modules

In the last chapter we used functions to decompose parts of your Shiny app into independent pieces. Functions work well for code that is either completely on the server side or completely on the client side. For code that spans both, i.e. whether the server code relies on specific structure in the UI, you'll need a new technique: modules.

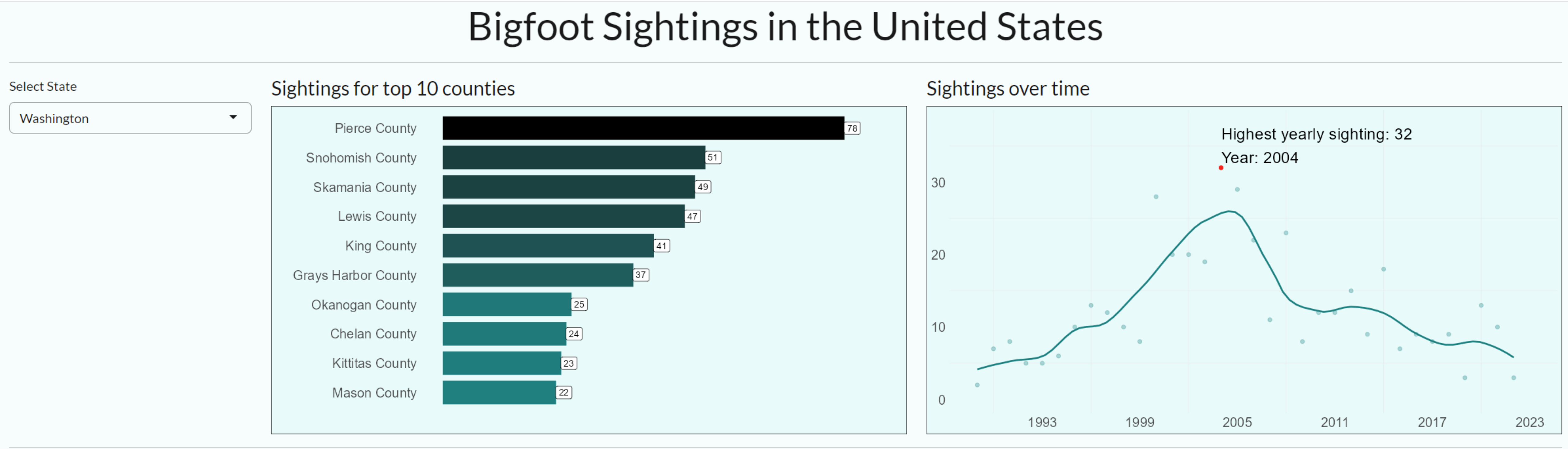
At the simplest level, a module is a pair of UI and server functions. The magic of modules comes because these functions are constructed in a special way that creates a "namespace". So far, when writing an app, the names (`id`s) of the controls are global: all parts of your server function can see all parts of your UI. Modules give you the ability to create controls that can only be seen from within the module. This is called a **namespace** because it creates "spaces" of "names" that are isolated from the rest of the app.

Shiny modules have two big advantages. Firstly, namespacing makes it easier to understand how your app works because you can write, analyse, and test individual components in isolation. Secondly, because modules are functions they help you reuse code; anything you can do with a function, you can do with a module.

```
library(shiny)
```

Mastering Shiny book by Hadley Wickham - https://mastering-shiny.org/

Let’s learn to write modules with a Shiny application

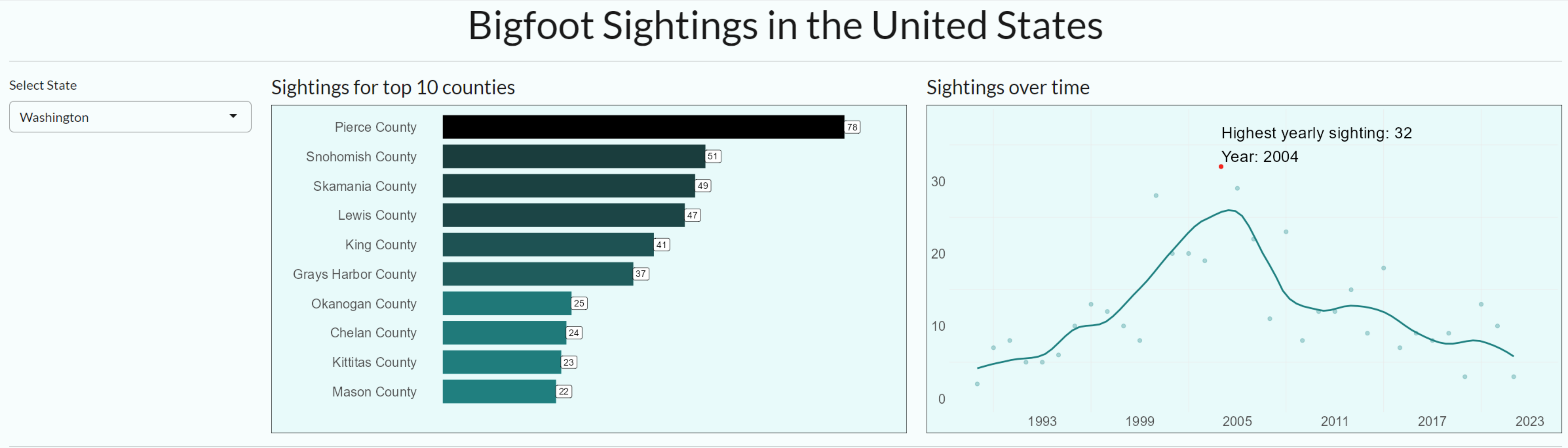


Bigfoot Sightings in the United States

Select State

Washington

Step 1 : Decompose your application components



Bigfoot Sightings in the United States

Select State

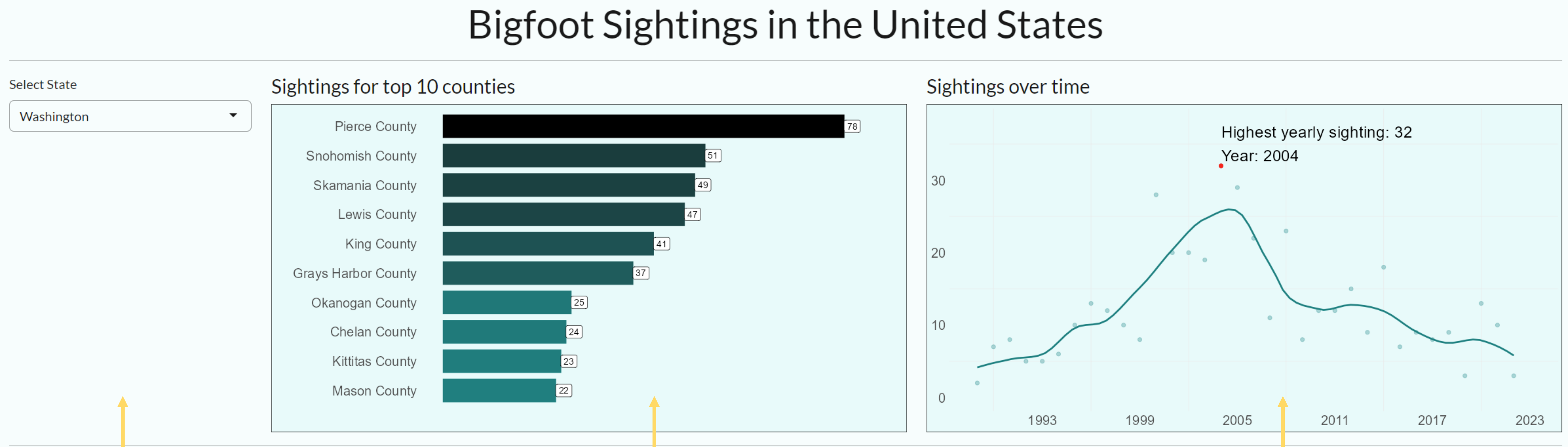
Washington

Sightings for top 10 counties

Sightings over time

Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/

Step 1 : Decompose your application components



Bigfoot Sightings in the United States

Select State

Washington

Sightings for top 10 counties

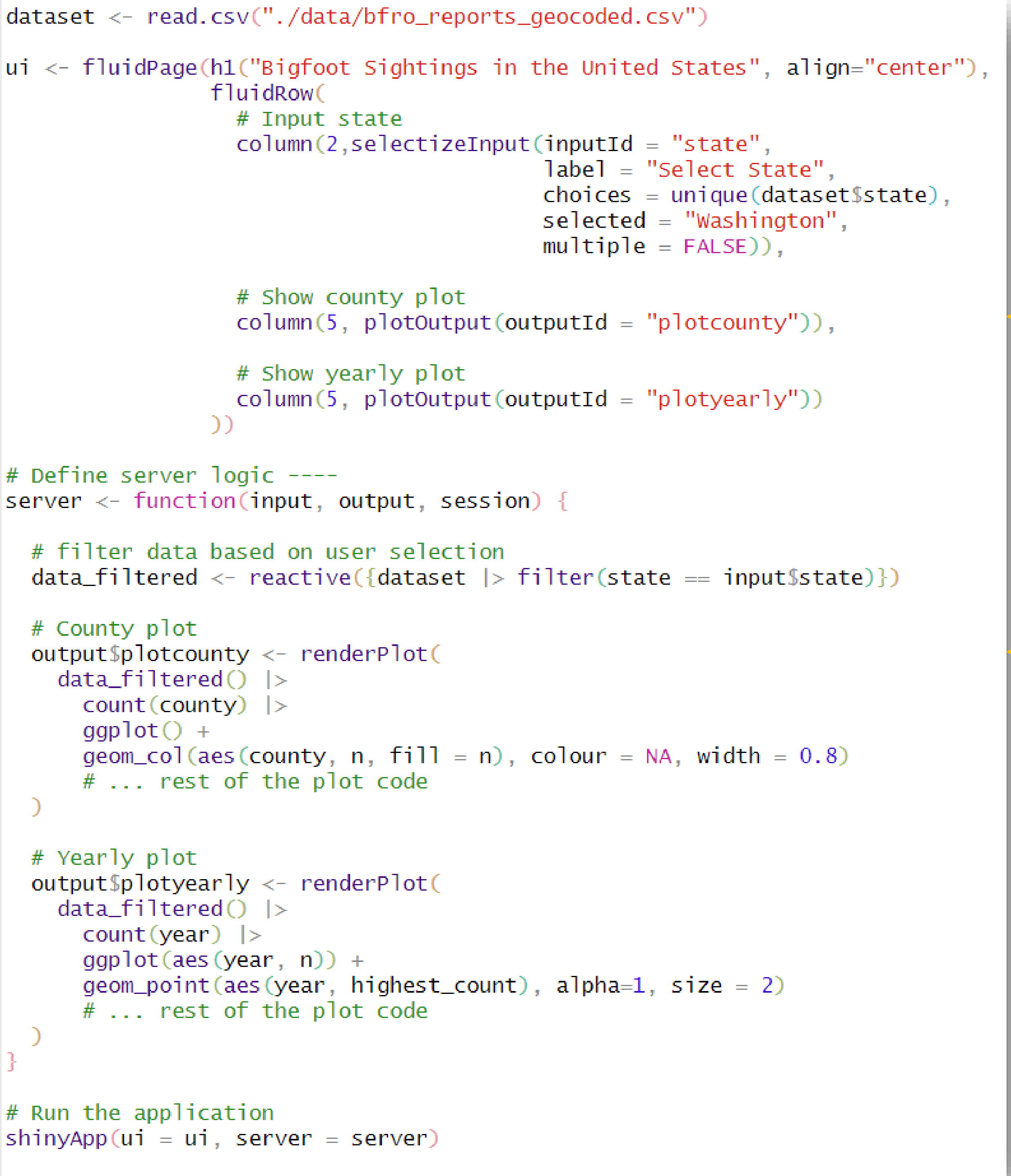
Sightings over time

Input for filtering data

County plot (on filtered data)

Yearly plot (on filtered data)

Main app.R file

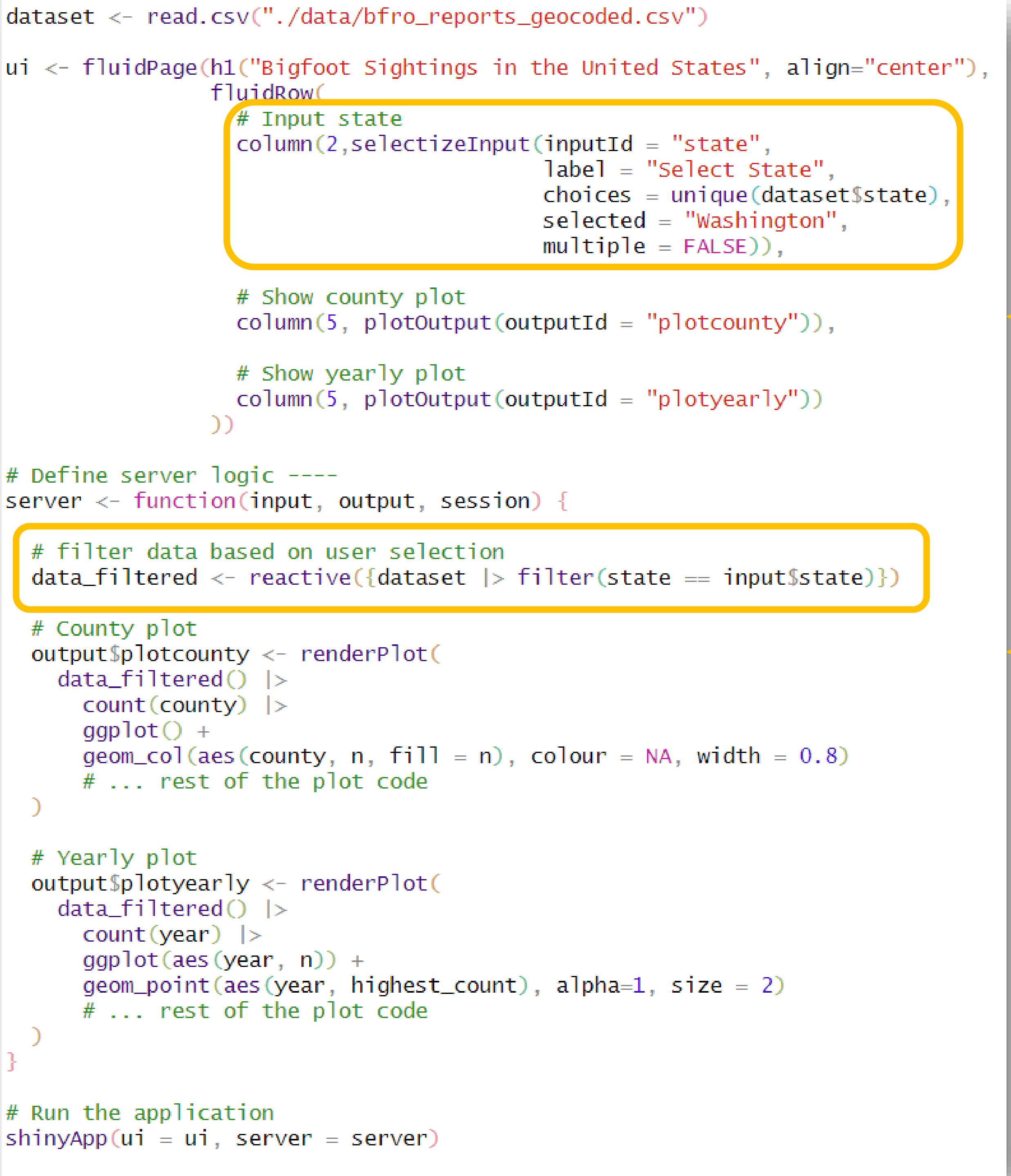


```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

3 ui elements

3 server elements

Main app.R file

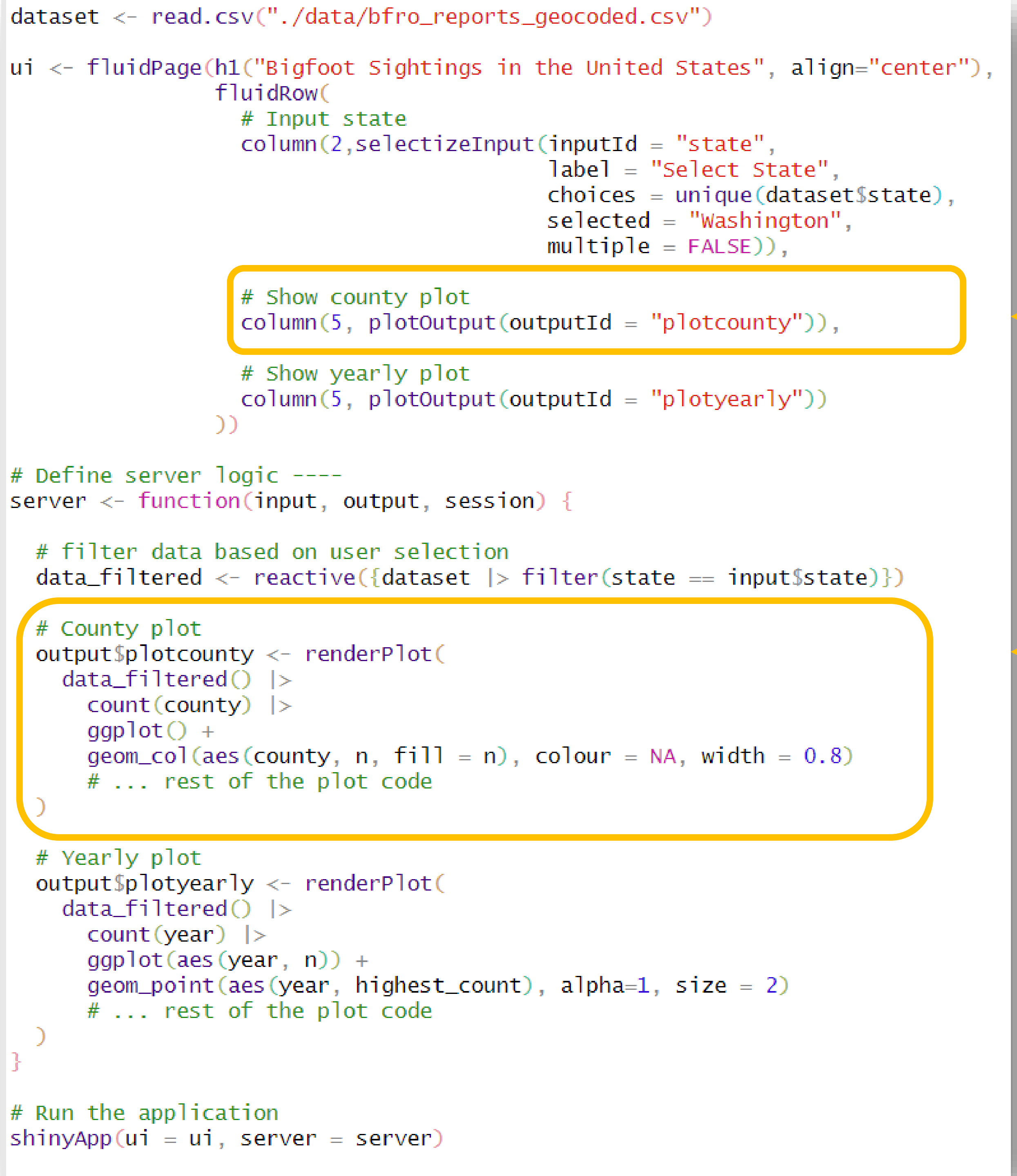


```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

3 ui elements

3 server elements

Main app.R file



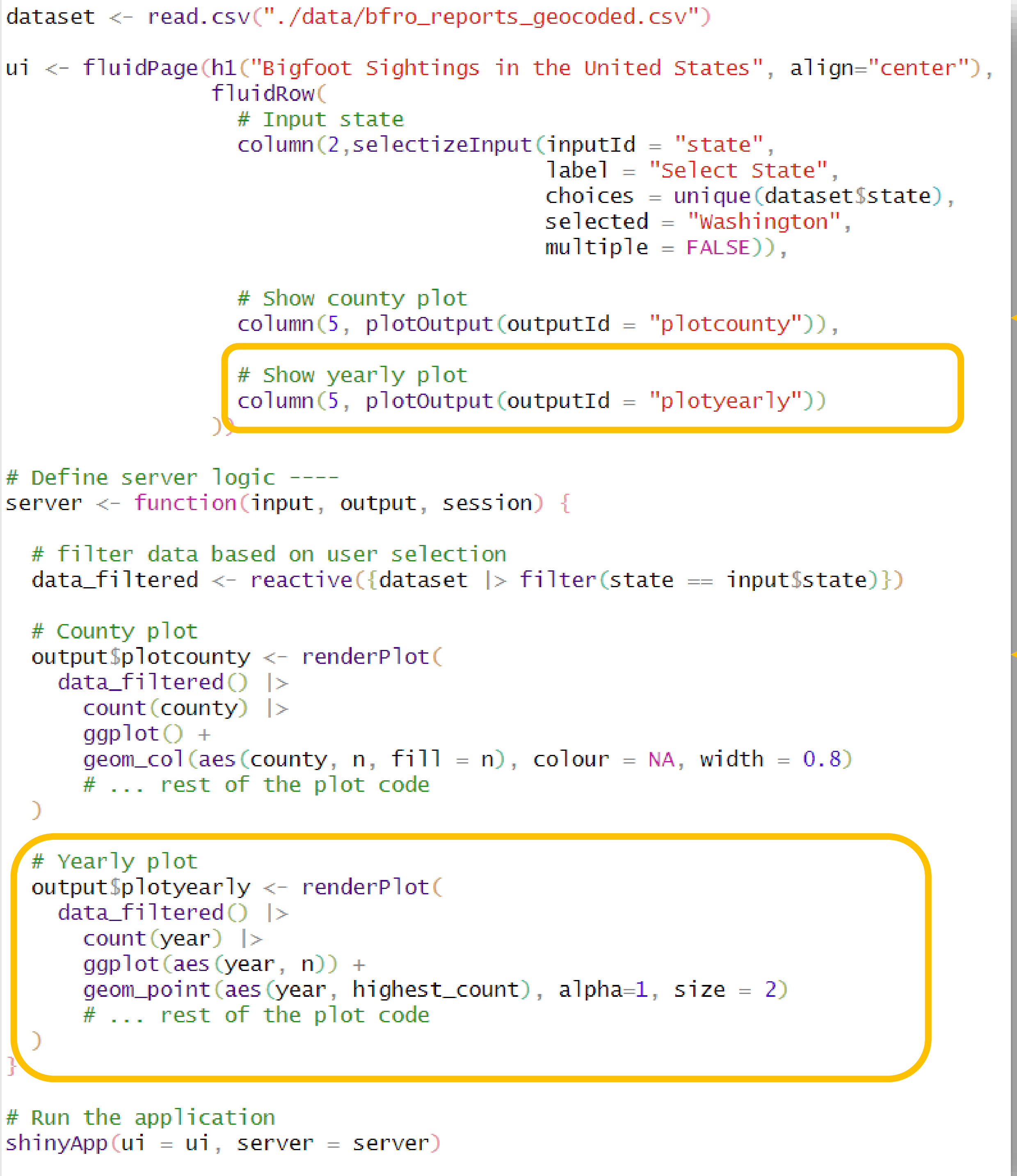
```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

3 ui elements

3 server elements

Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/

Main app.R file



```
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

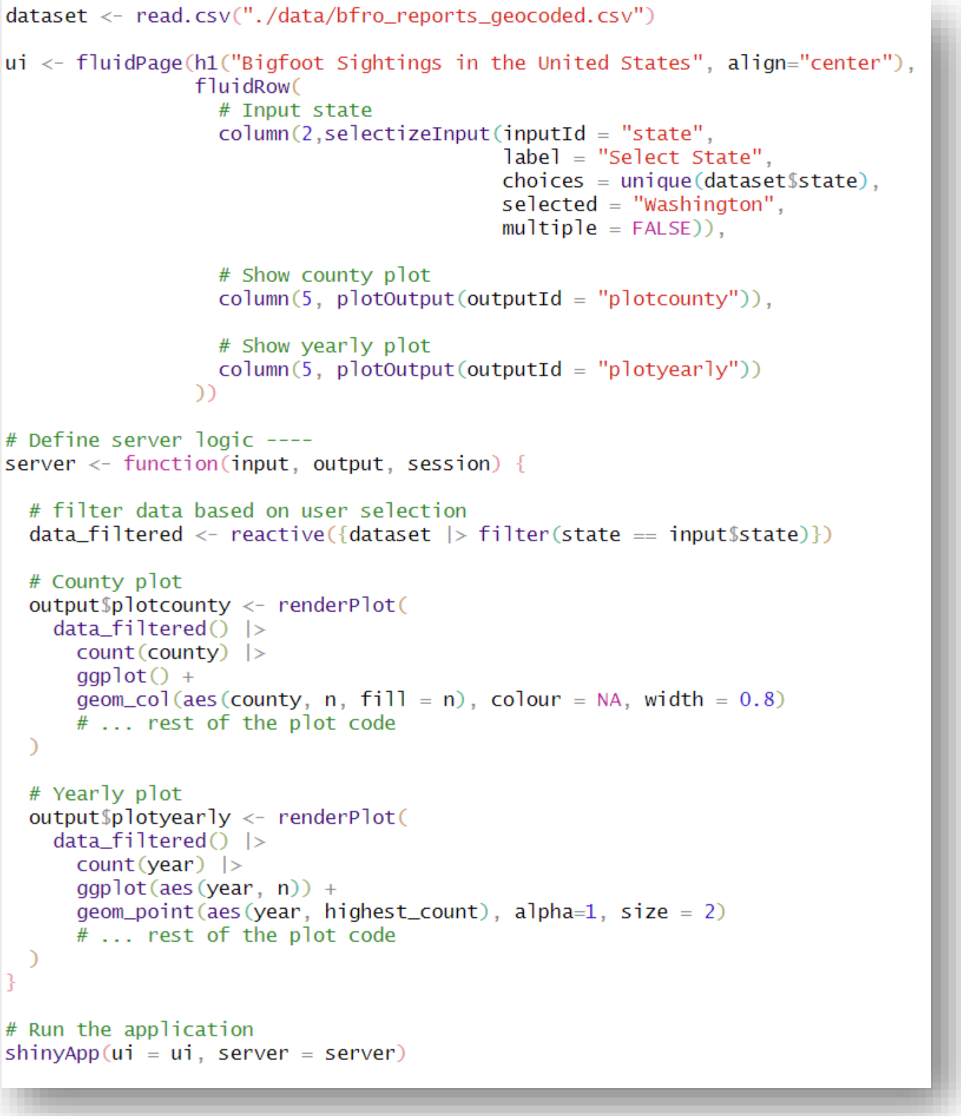
3 ui elements

3 server elements

Step 2 : Create modules (baby shiny apps) for the distributed components

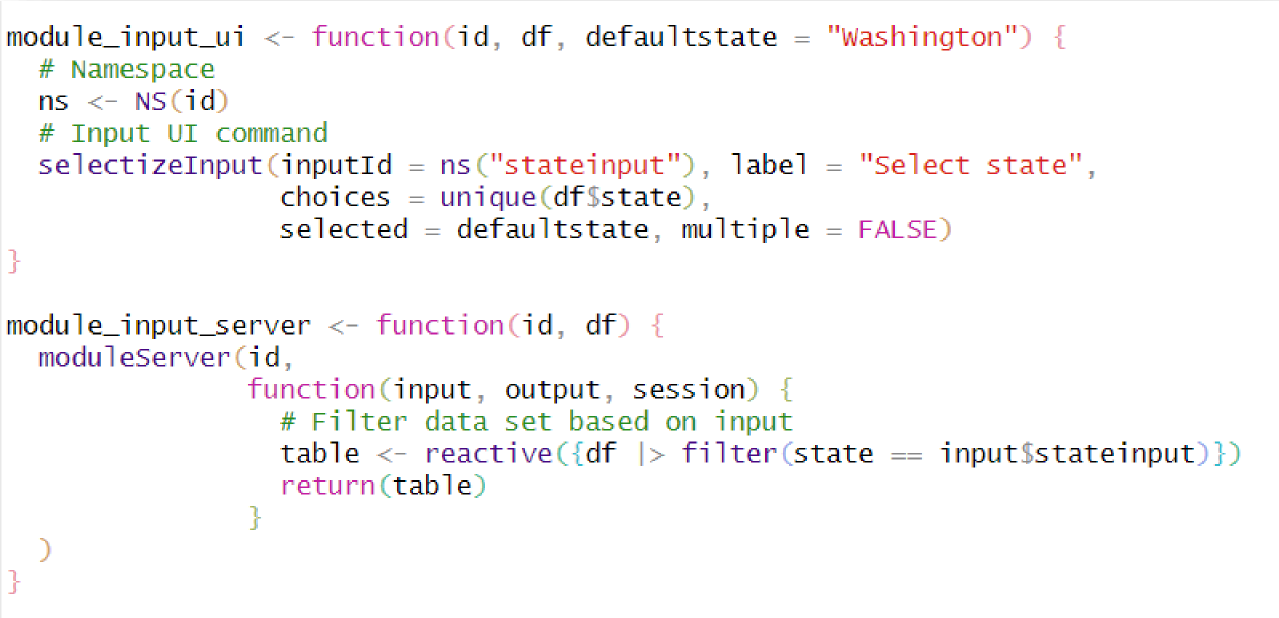
- Module 1: Module_input.R
  - Ui: Create state input ui
  - Server: Filter data based on state input
- Module 2: Module_countyplot.R
  - Ui: Create county plot ui
  - Server: Generate the county plot with ggplot code
- Module 3: Module_yearlyplot.R
  - Ui: Create yearly plot ui
  - Server: Generate the yearly plot with ggplot code

Original app.R file



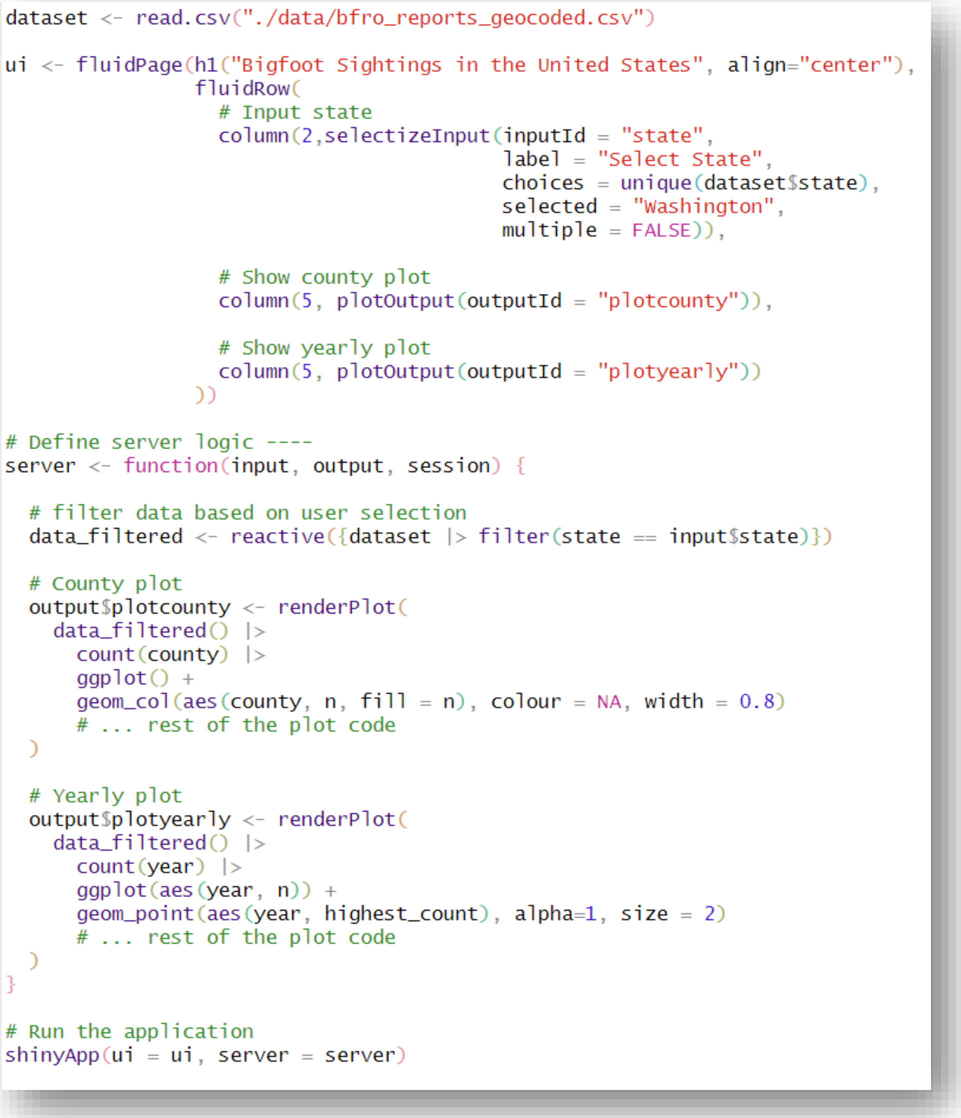
```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Module 1: Module_input.R



```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

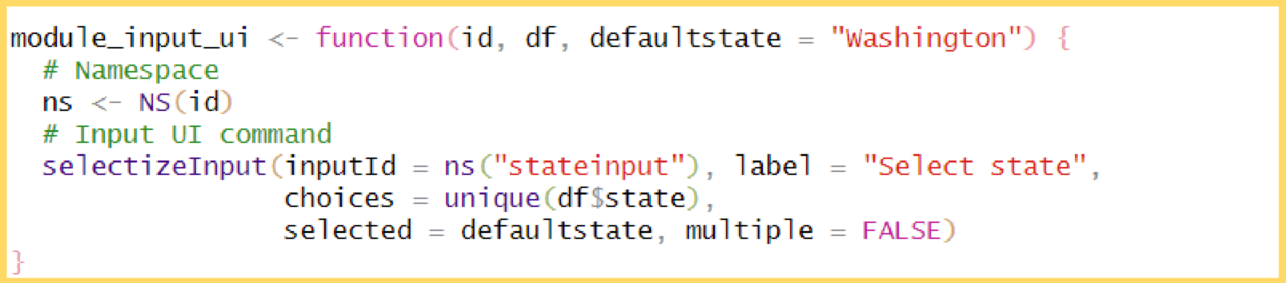
Original app.R file



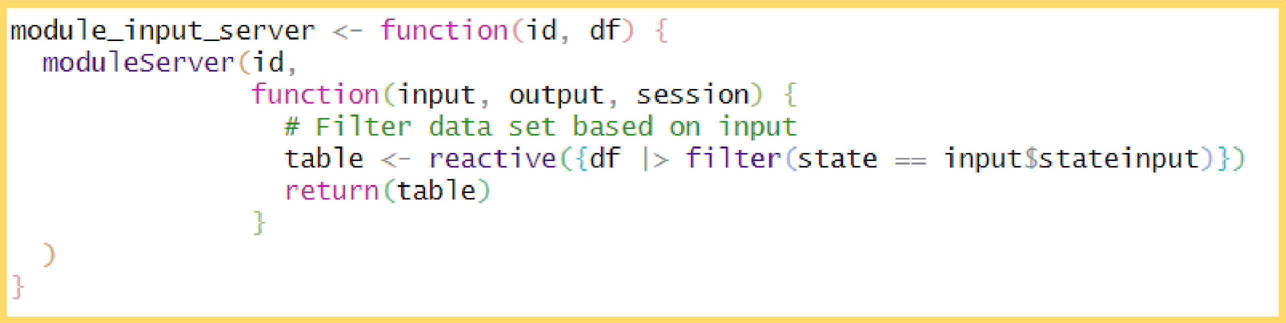
```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),


                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Module 1: Module_input.R

ui and server components are written as two separate modules, just like a shiny app.

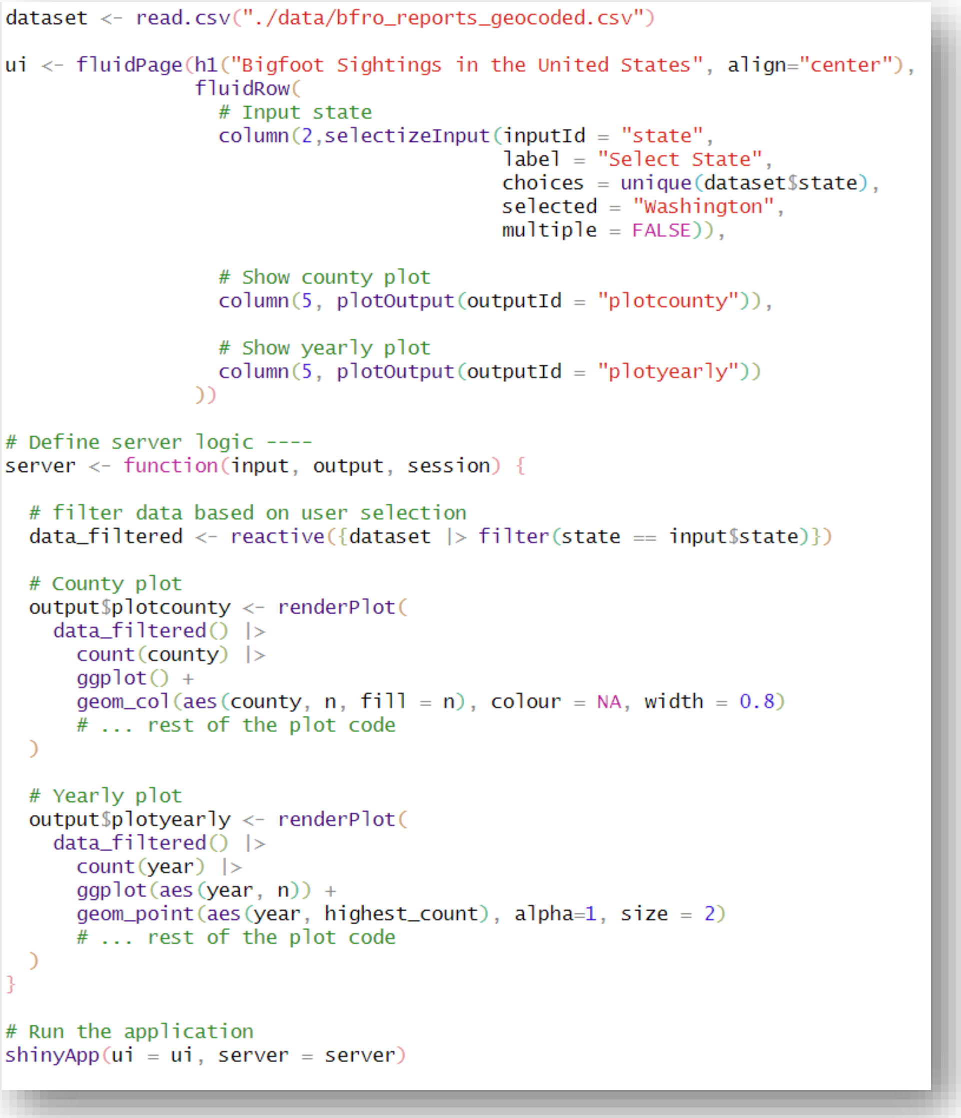


```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}
```



```r
module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

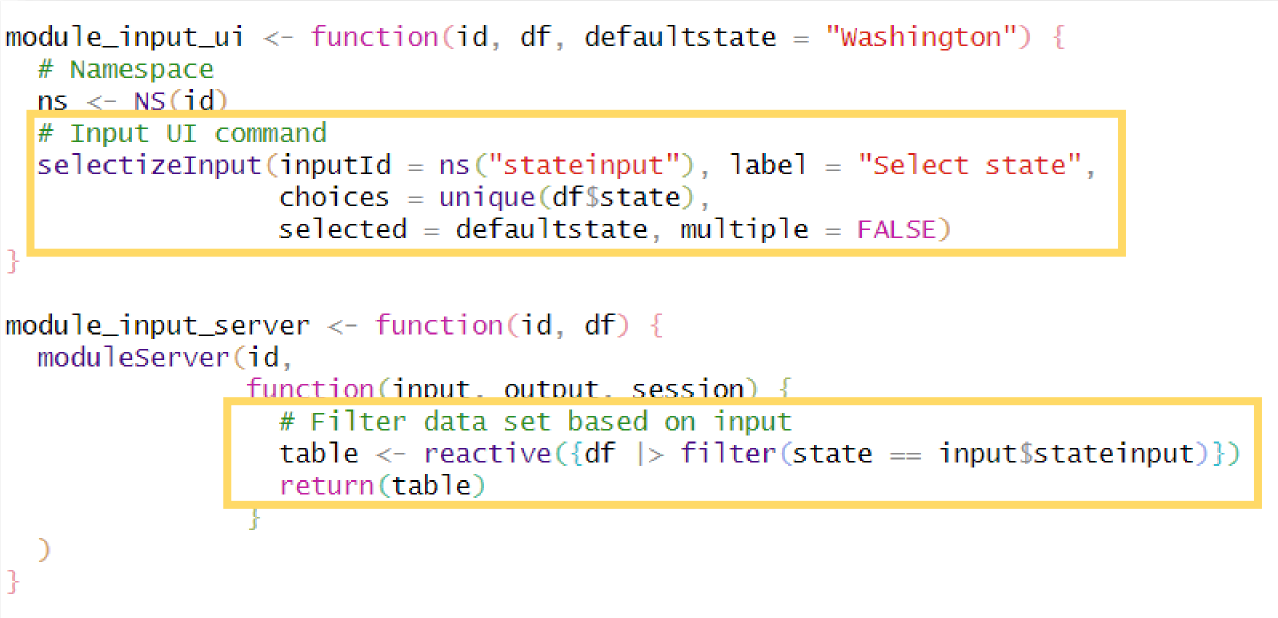
Original app.R file



```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

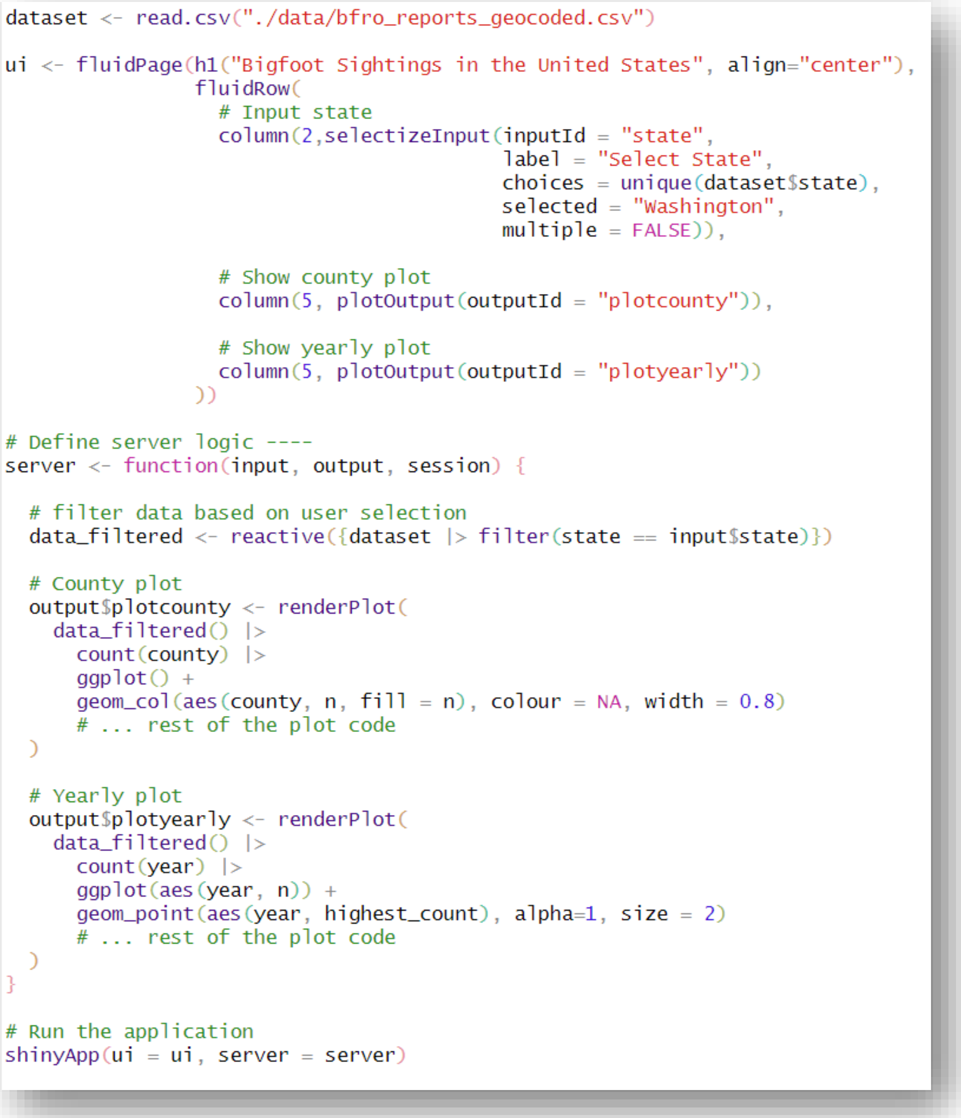
Module 1: Module_input.R

In this module we want to ask user to input "state" and then filter data based on the user input



```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

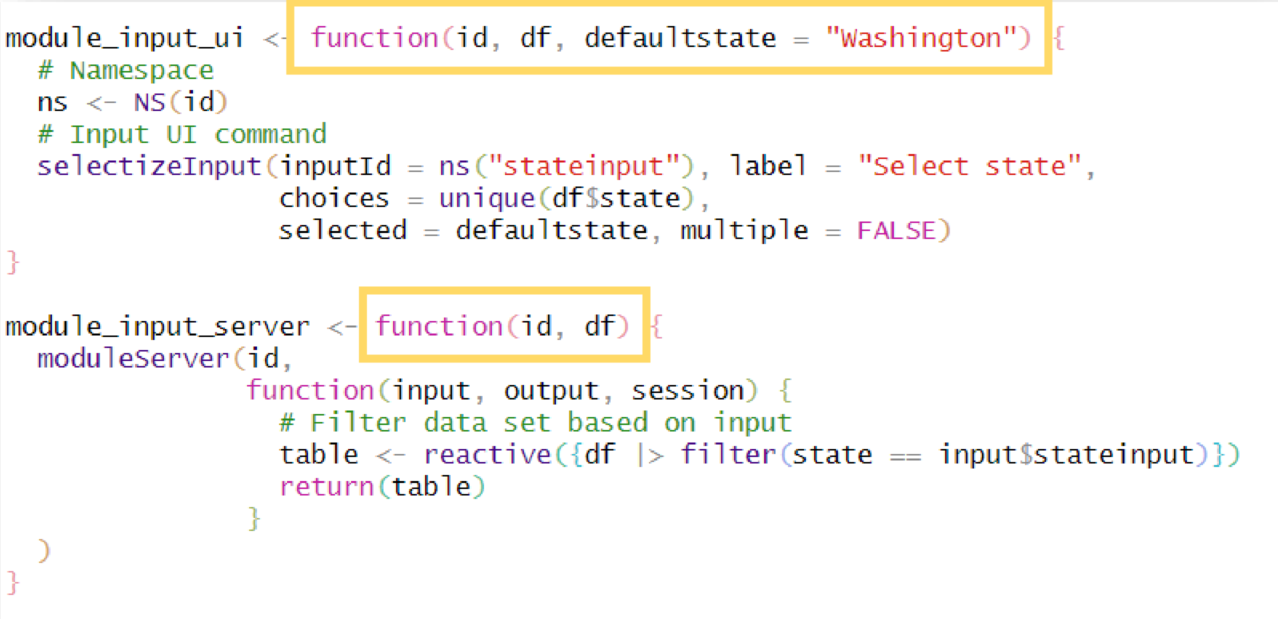
Original app.R file



```
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),


                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

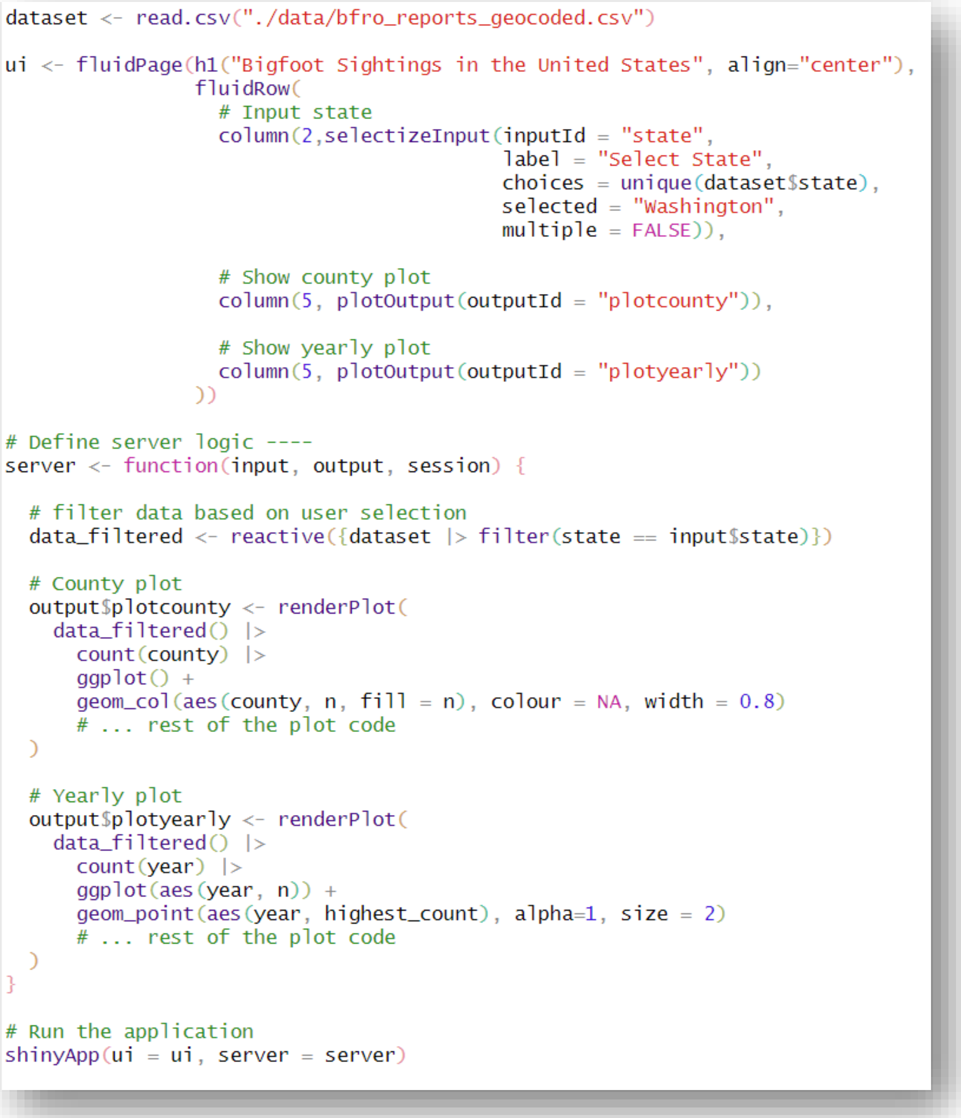
Module 1: Module_input.R

We create the ui and server as functions that can be called from the main app.R file



```
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

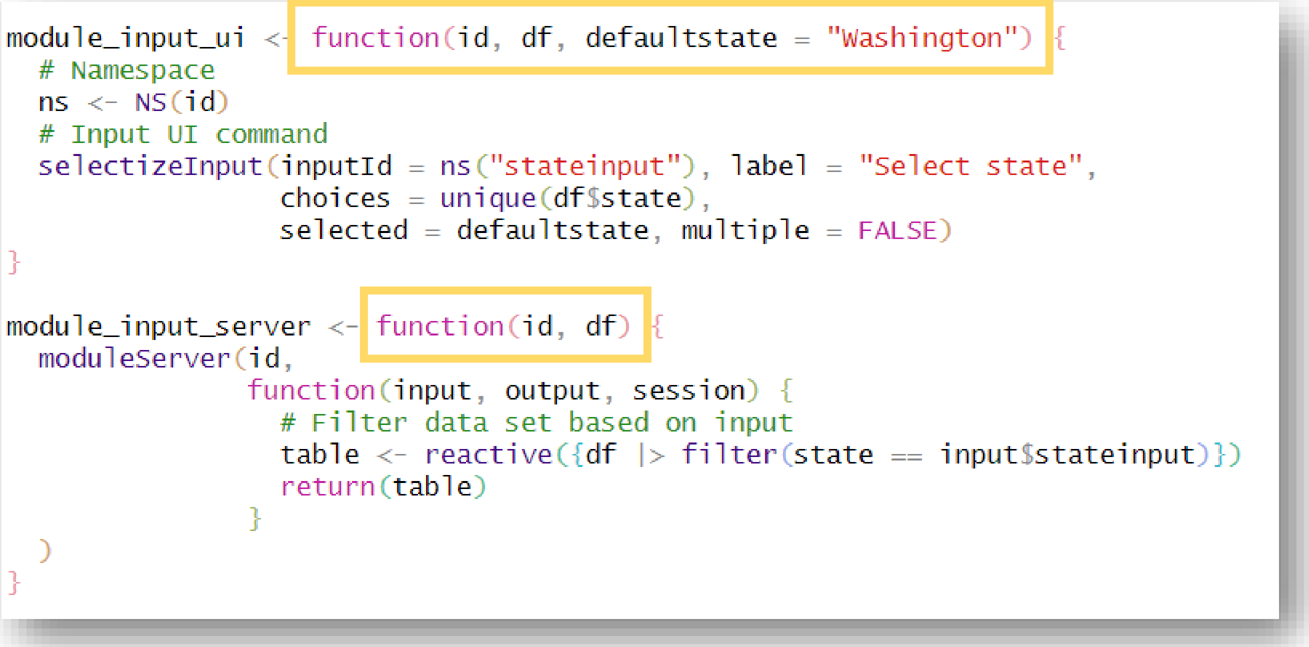
Original app.R file



```
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Module 1: Module_input.R

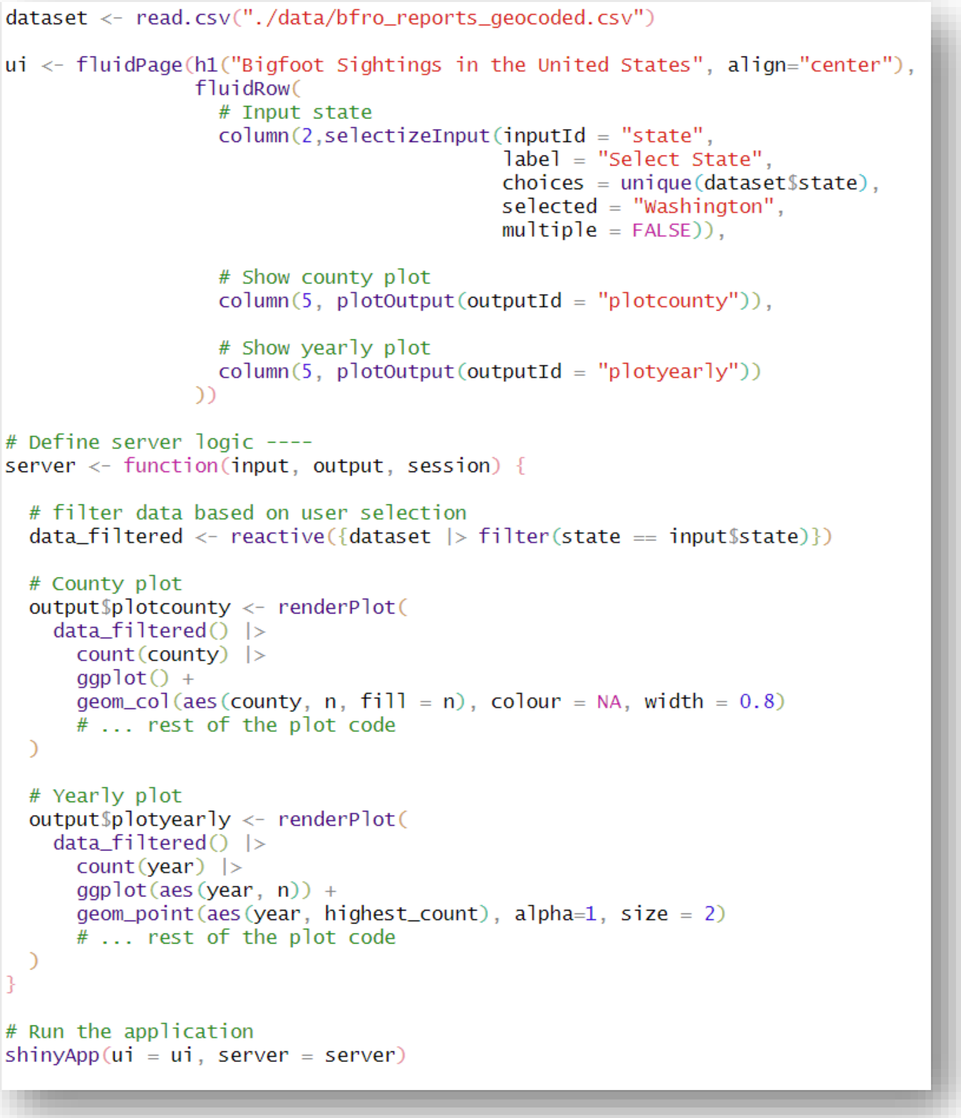
ID serves as a unique identifier for each instance of the module within the application, allowing multiple instances of the same module to coexist and be independently controlled.



```
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/

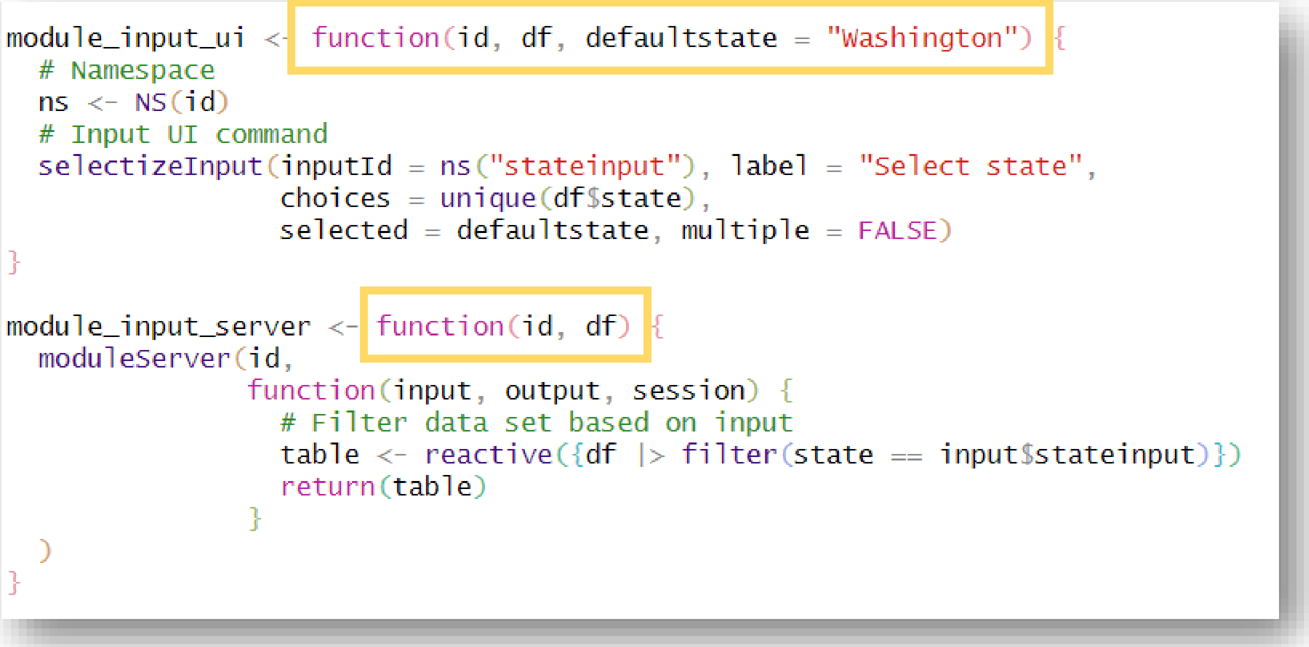
Original app.R file



```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Module 1: Module_input.R

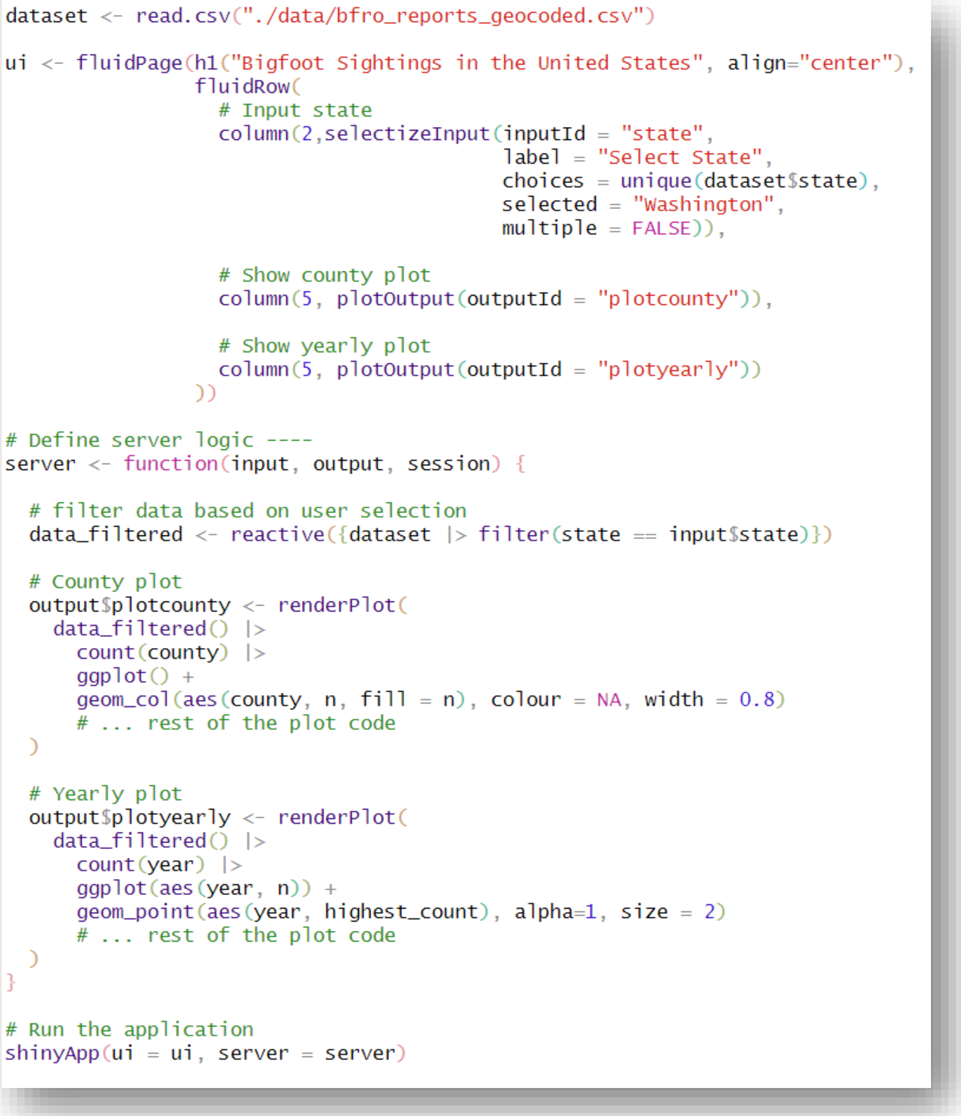
ID serves as a unique identifier for each instance of the module within the application, allowing multiple instances of the same module to coexist and be independently controlled.



```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

ID is instantiated when the module function is called from the main app. Each ui and server call have the same unique id connecting the two.

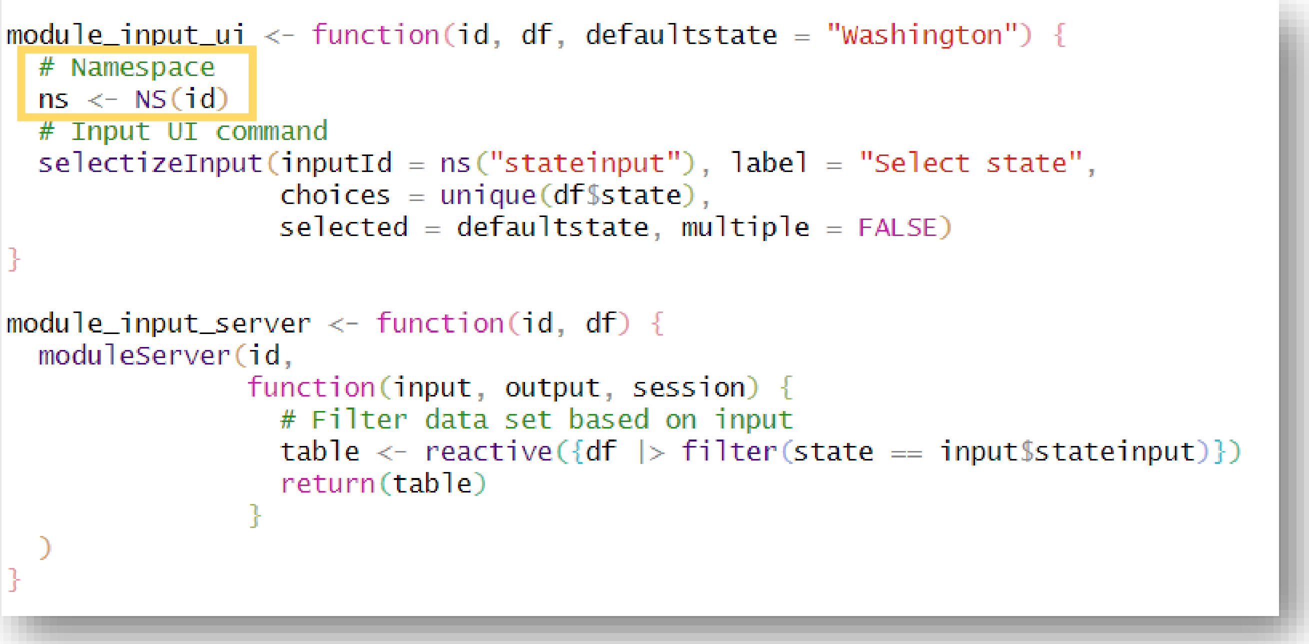
Original app.R file



```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

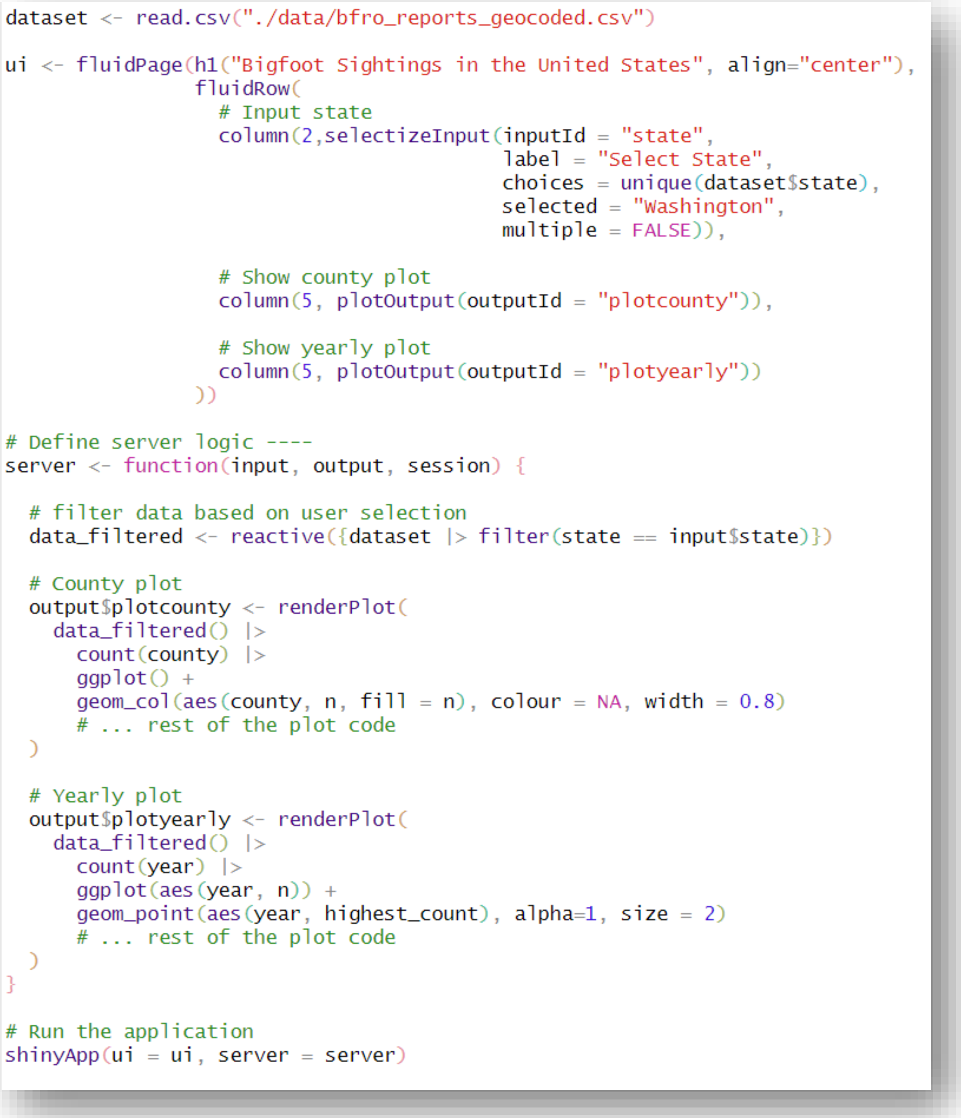
Module 1: Module_input.R

A namespace is like a separate room that contains all the functions and variables specific to that module, preventing naming conflicts and keeping things organized



```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

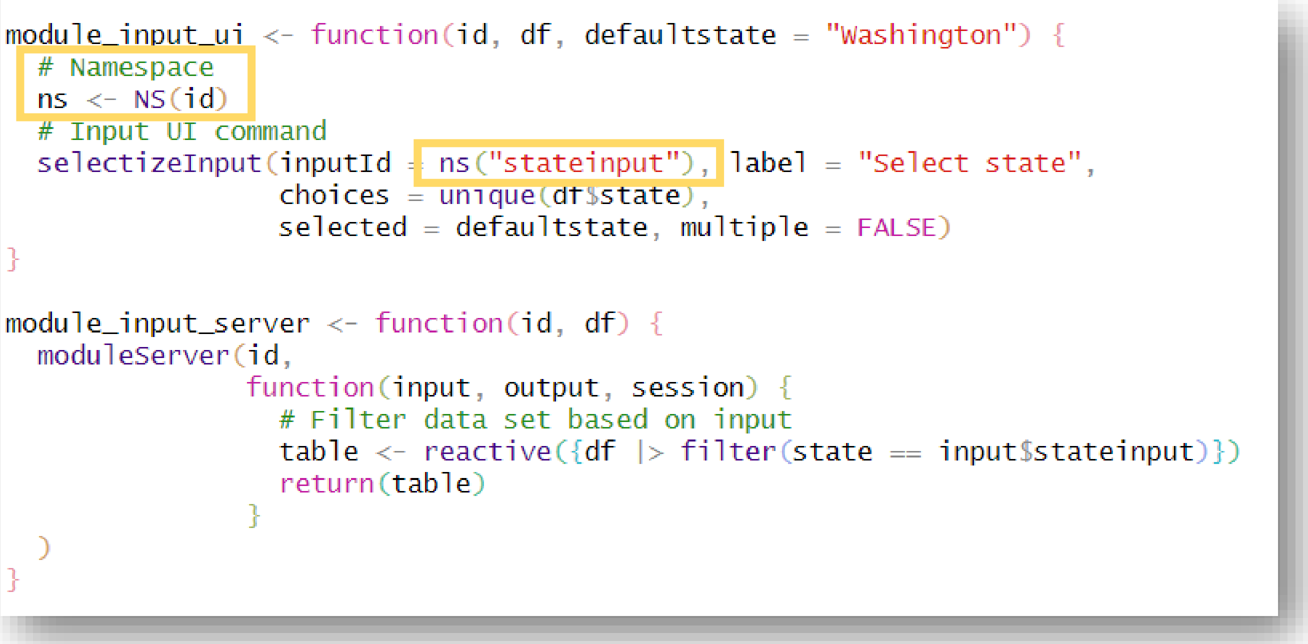
Original app.R file



```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),


                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Module 1: Module_input.R

A namespace is like a separate room that contains all the functions and variables specific to that module, preventing naming conflicts and keeping things organized



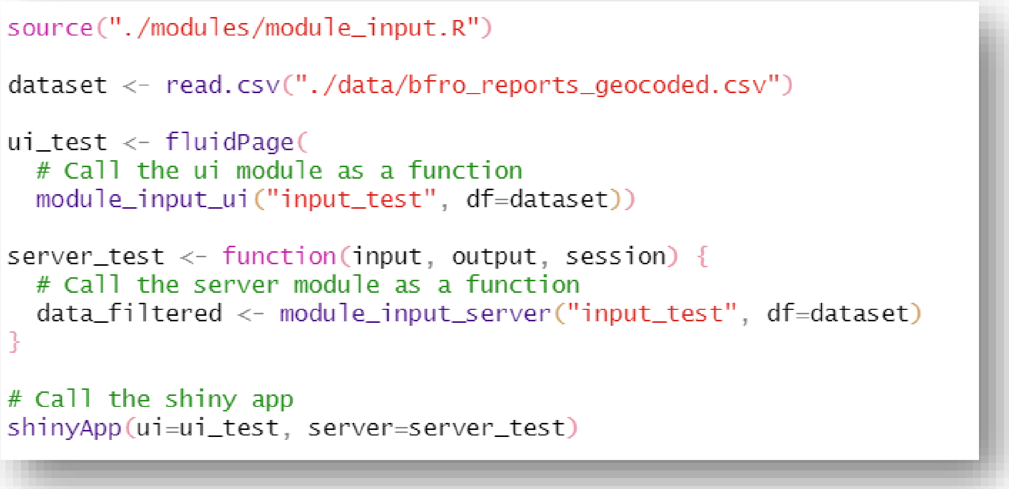
```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

Each input name is encapsulated in ns() to ensure that it is properly namespaced and avoid conflicts with input names from other module calls in the main application

Step 3: Test your module

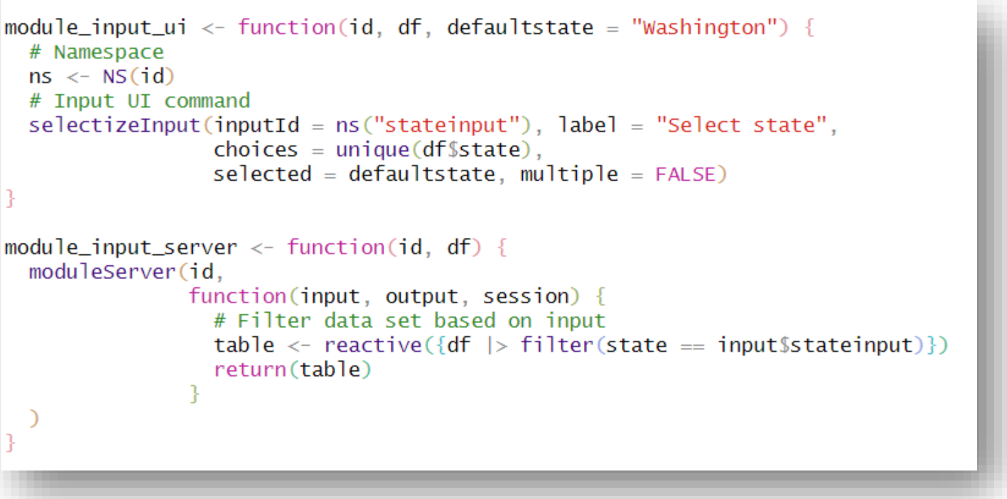
Call the module function from the main app.R file

Test app.R



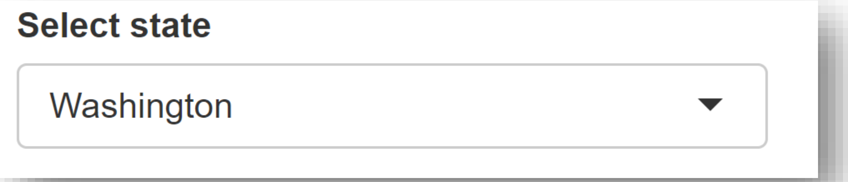
```r
source("./modules/module_input.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui_test <- fluidPage(
  # Call the ui module as a function
  module_input_ui("input_test", df=dataset))

server_test <- function(input, output, session) {
  # Call the server module as a function
  data_filtered <- module_input_server("input_test", df=dataset)
}

# Call the shiny app
shinyApp(ui=ui_test, server=server_test)
```

Module 1: Module_input.R



```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

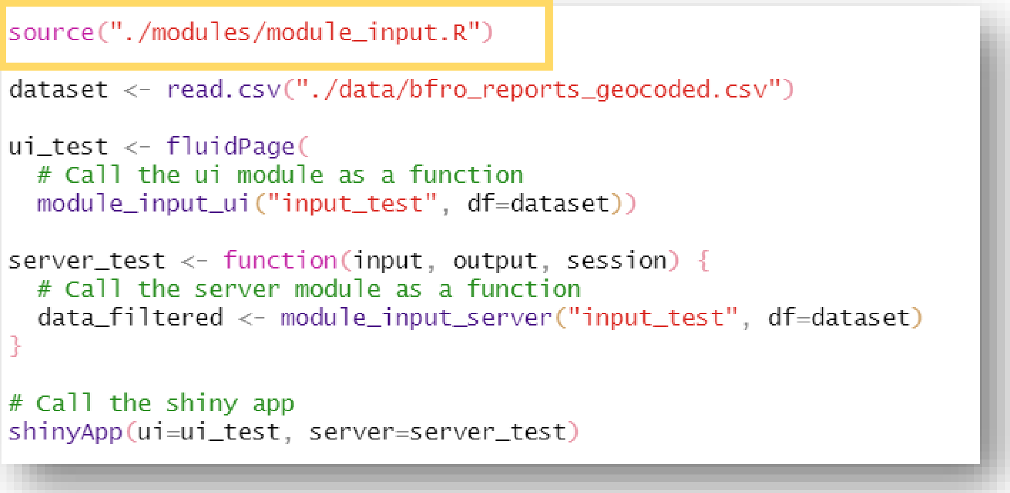
Shiny App output



Step 3: Test your module

Call the module function from the main app.R file

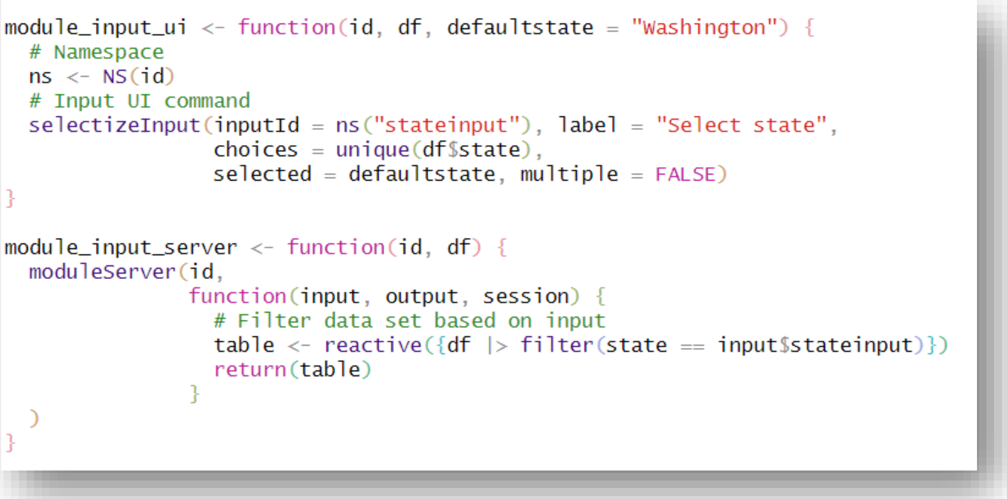
Test app.R



```r
source("./modules/module_input.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui_test <- fluidPage(
  # Call the ui module as a function
  module_input_ui("input_test", df=dataset))

server_test <- function(input, output, session) {
  # Call the server module as a function
  data_filtered <- module_input_server("input_test", df=dataset)
}

# Call the shiny app
shinyApp(ui=ui_test, server=server_test)
```

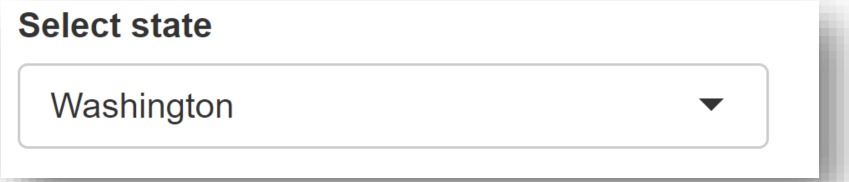
Source your module_input.R file

Module 1: Module_input.R



```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

Shiny App output



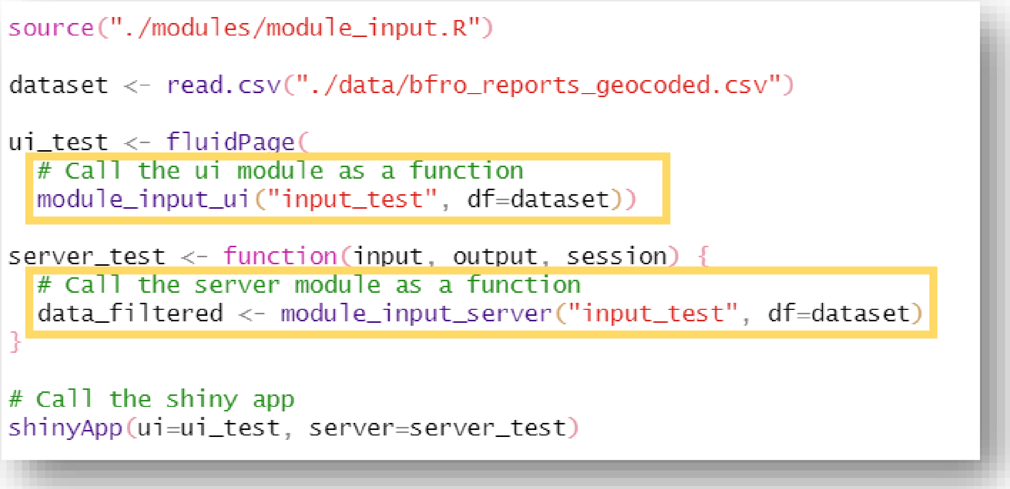
Select state

Washington ▾

Step 3: Test your module

Call the module function from the main app.R file

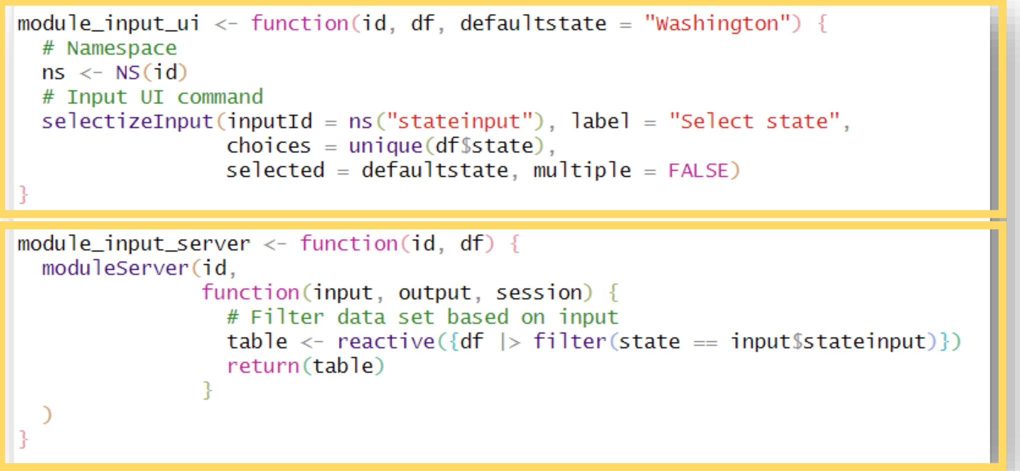
Test app.R



```r
source("./modules/module_input.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui_test <- fluidPage(
  # Call the ui module as a function
  module_input_ui("input_test", df=dataset))

server_test <- function(input, output, session) {
  # Call the server module as a function
  data_filtered <- module_input_server("input_test", df=dataset)
}

# Call the shiny app
shinyApp(ui=ui_test, server=server_test)
```

Call the ui module and server module functions from the main app ui and server

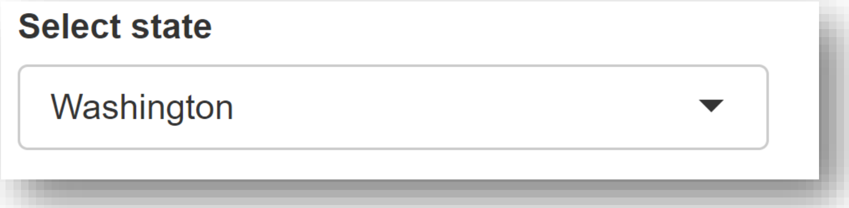
Module 1: Module_input.R



```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}
```

```r
module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

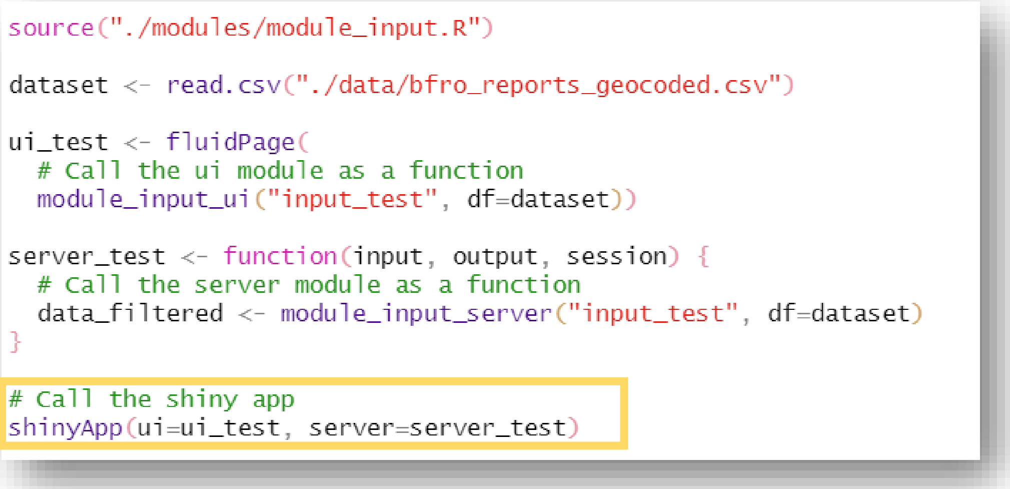
Shiny App output



Step 3: Test your module

Call the module function from the main app.R file

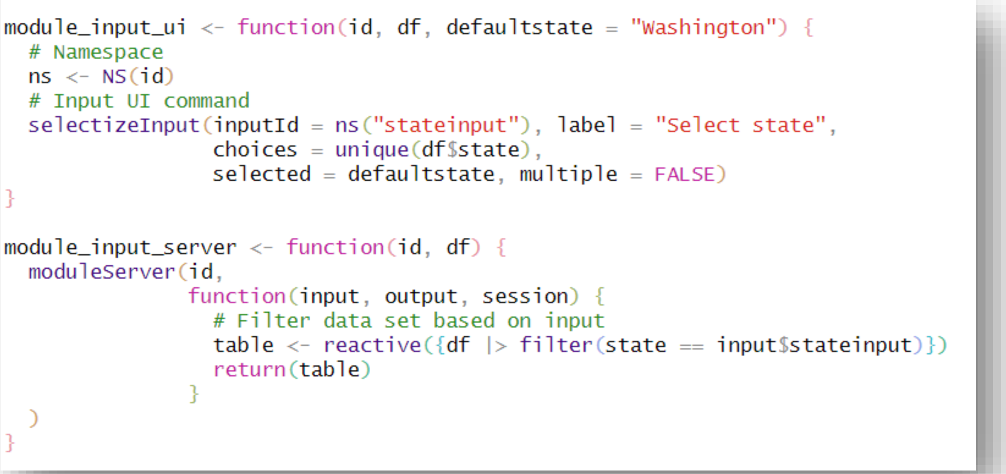
Test app.R



```
source("./modules/module_input.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui_test <- fluidPage(
  # Call the ui module as a function
  module_input_ui("input_test", df=dataset))

server_test <- function(input, output, session) {
  # Call the server module as a function
  data_filtered <- module_input_server("input_test", df=dataset)
}

# Call the shiny app
shinyApp(ui=ui_test, server=server_test)
```

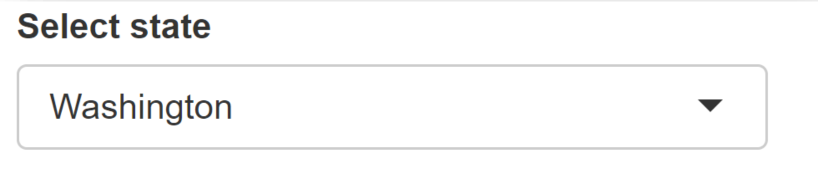
Call the shiny app command

Module 1: Module_input.R



```
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

Shiny App output



Select state

Washington ▼

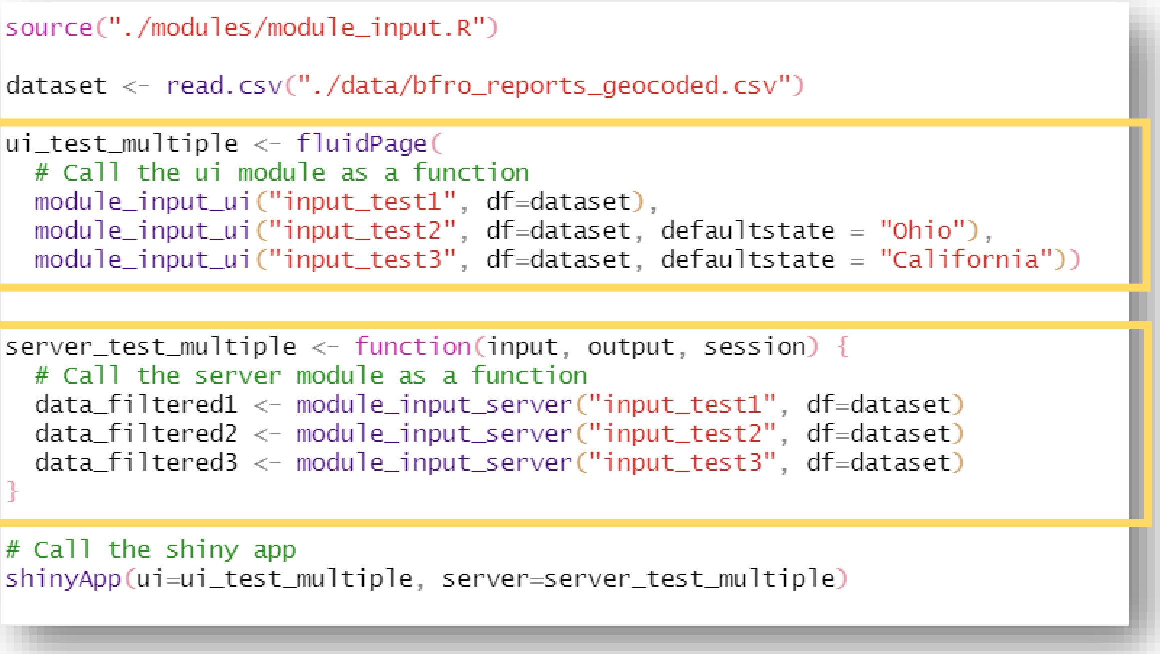
Shiny app is created

Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/

Handling multiple module calls

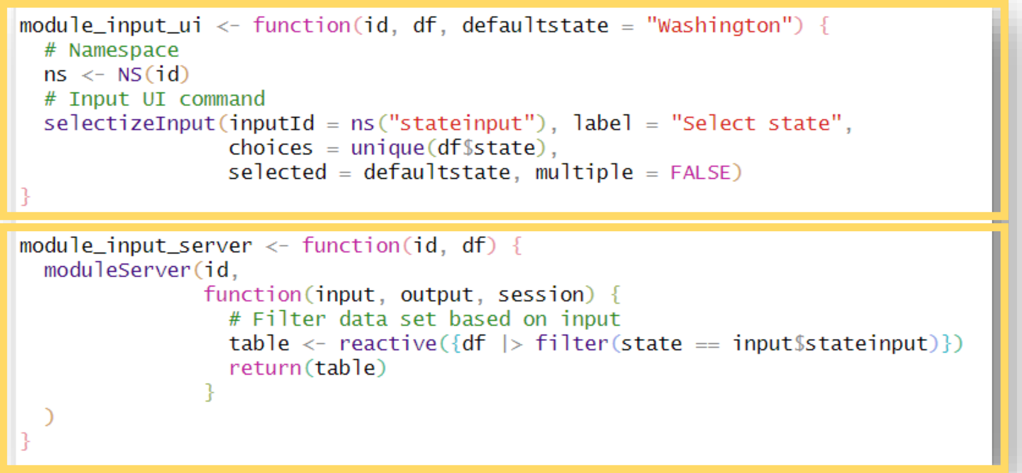
Call the module function from the main app.R file multiple times with separate unique IDs – it is that simple!!

Test app.R



```r
source("./modules/module_input.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui_test_multiple <- fluidPage(
  # Call the ui module as a function
  module_input_ui("input_test1", df=dataset),
  module_input_ui("input_test2", df=dataset, defaultstate = "Ohio"),
  module_input_ui("input_test3", df=dataset, defaultstate = "California"))

server_test_multiple <- function(input, output, session) {
  # Call the server module as a function
  data_filtered1 <- module_input_server("input_test1", df=dataset)
  data_filtered2 <- module_input_server("input_test2", df=dataset)
  data_filtered3 <- module_input_server("input_test3", df=dataset)
}

# Call the shiny app
shinyApp(ui=ui_test_multiple, server=server_test_multiple)
```

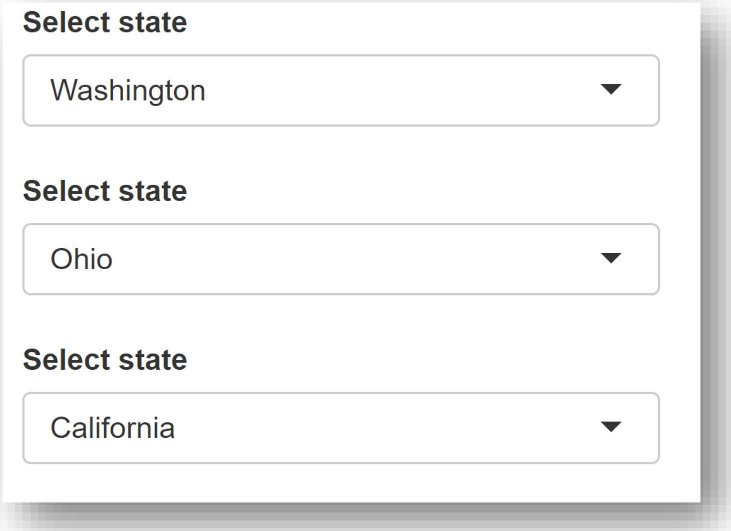
Module 1: Module_input.R



```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}
```

```r
module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

Shiny App output



Select state

Washington

Select state

Ohio

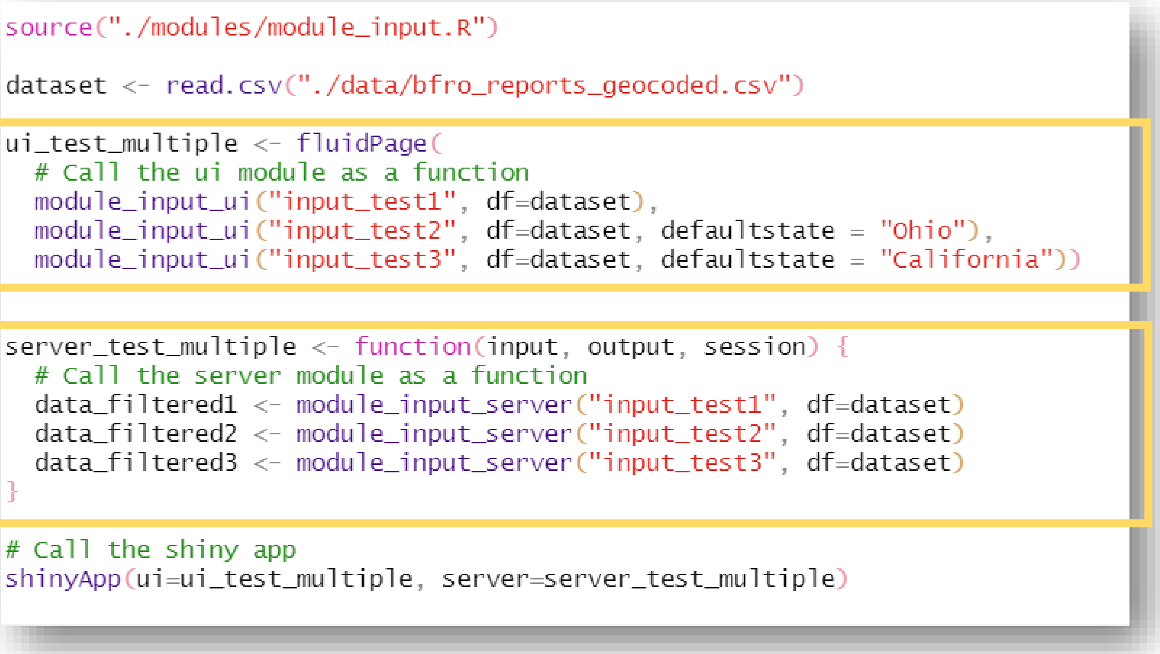
Select state

California

Handling multiple module calls

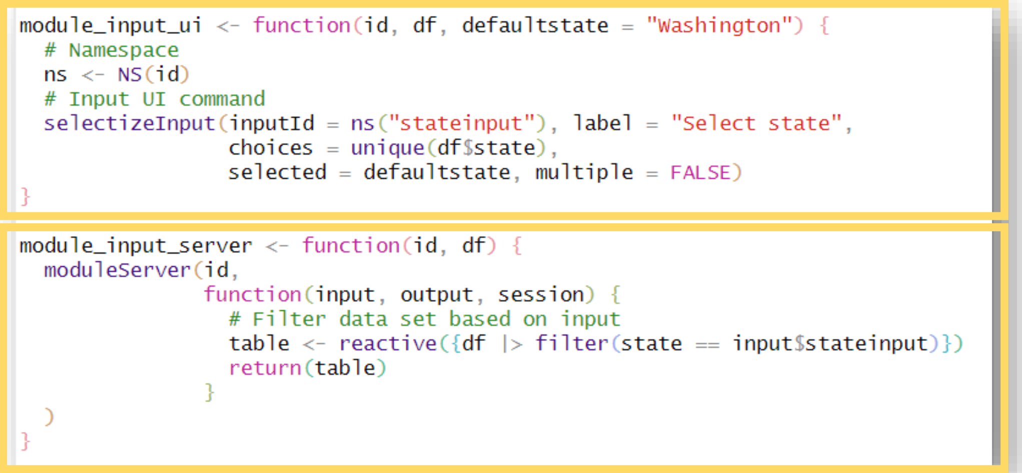
Call the module function from the main app.R file multiple times with separate unique IDs – it is that simple!!

Test app.R



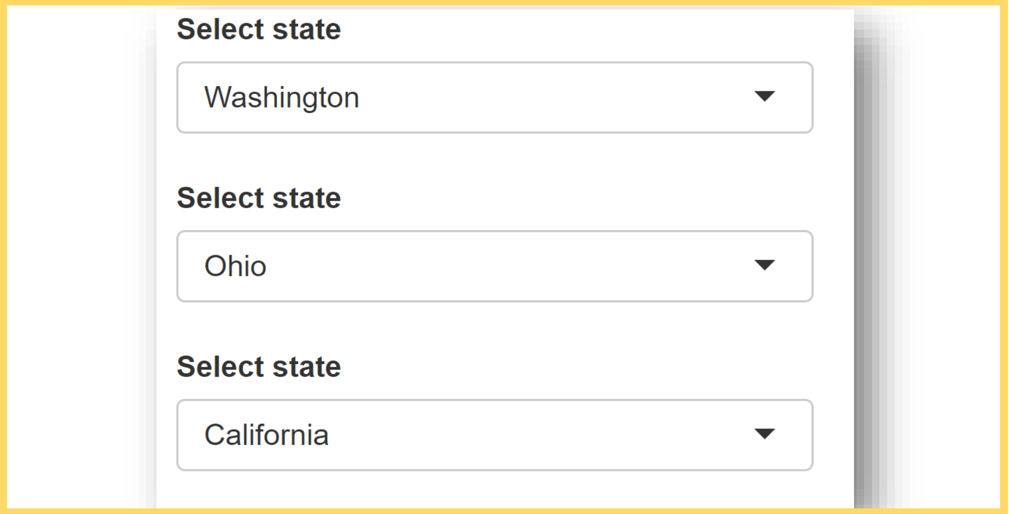
```r
source("./modules/module_input.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui_test_multiple <- fluidPage(
  # Call the ui module as a function
  module_input_ui("input_test1", df=dataset),
  module_input_ui("input_test2", df=dataset, defaultstate = "Ohio"),
  module_input_ui("input_test3", df=dataset, defaultstate = "California"))

server_test_multiple <- function(input, output, session) {
  # Call the server module as a function
  data_filtered1 <- module_input_server("input_test1", df=dataset)
  data_filtered2 <- module_input_server("input_test2", df=dataset)
  data_filtered3 <- module_input_server("input_test3", df=dataset)
}

# Call the shiny app
shinyApp(ui=ui_test_multiple, server=server_test_multiple)
```

Module 1: Module_input.R



```r
module_input_ui <- function(id, df, defaultstate = "Washington") {
  # Namespace
  ns <- NS(id)
  # Input UI command
  selectizeInput(inputId = ns("stateinput"), label = "Select state",
                 choices = unique(df$state),
                 selected = defaultstate, multiple = FALSE)
}

module_input_server <- function(id, df) {
  moduleServer(id,
               function(input, output, session) {
                 # Filter data set based on input
                 table <- reactive({df |> filter(state == input$stateinput)})
                 return(table)
               }
  )
}
```

Shiny App output



Select state

Washington ▼

Select state

Ohio ▼

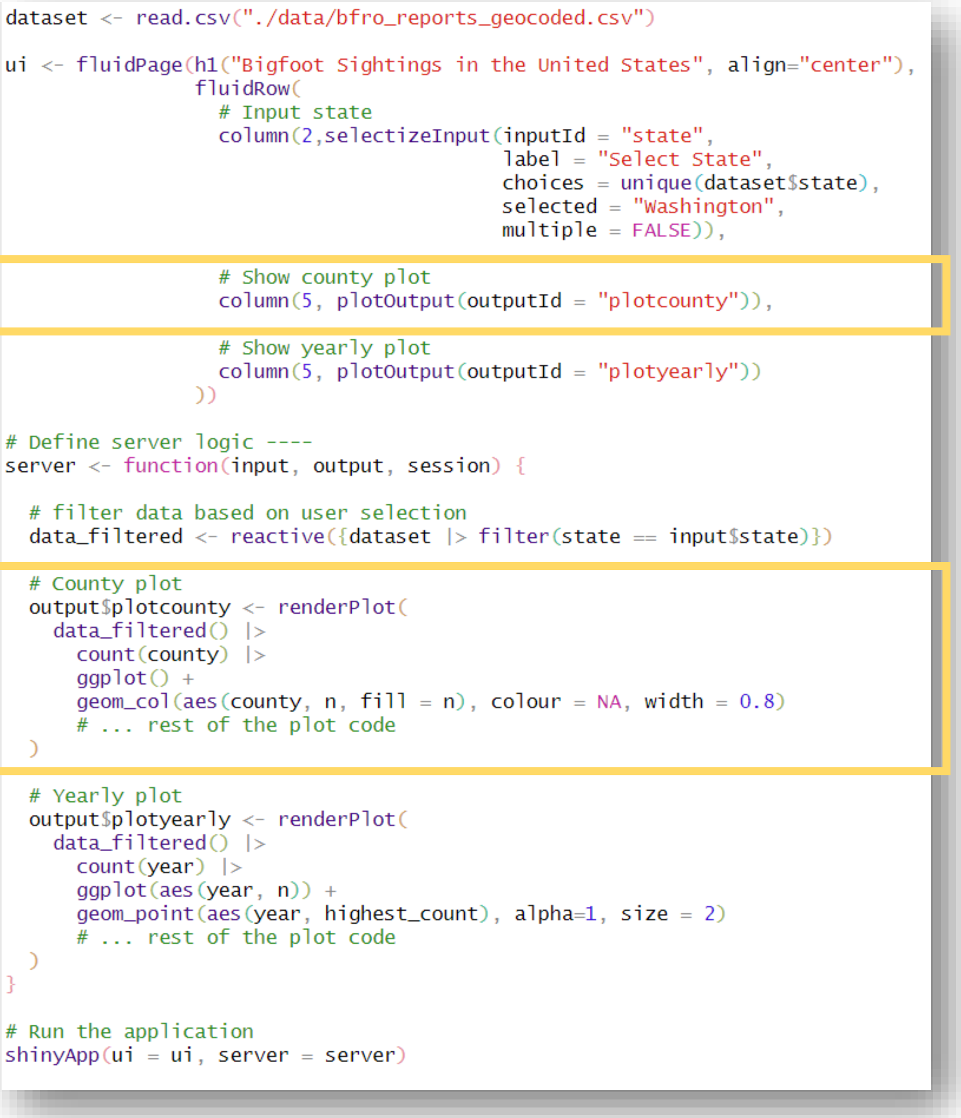
Select state

California ▼

Step 4: Replicate modularization for other decomposed components

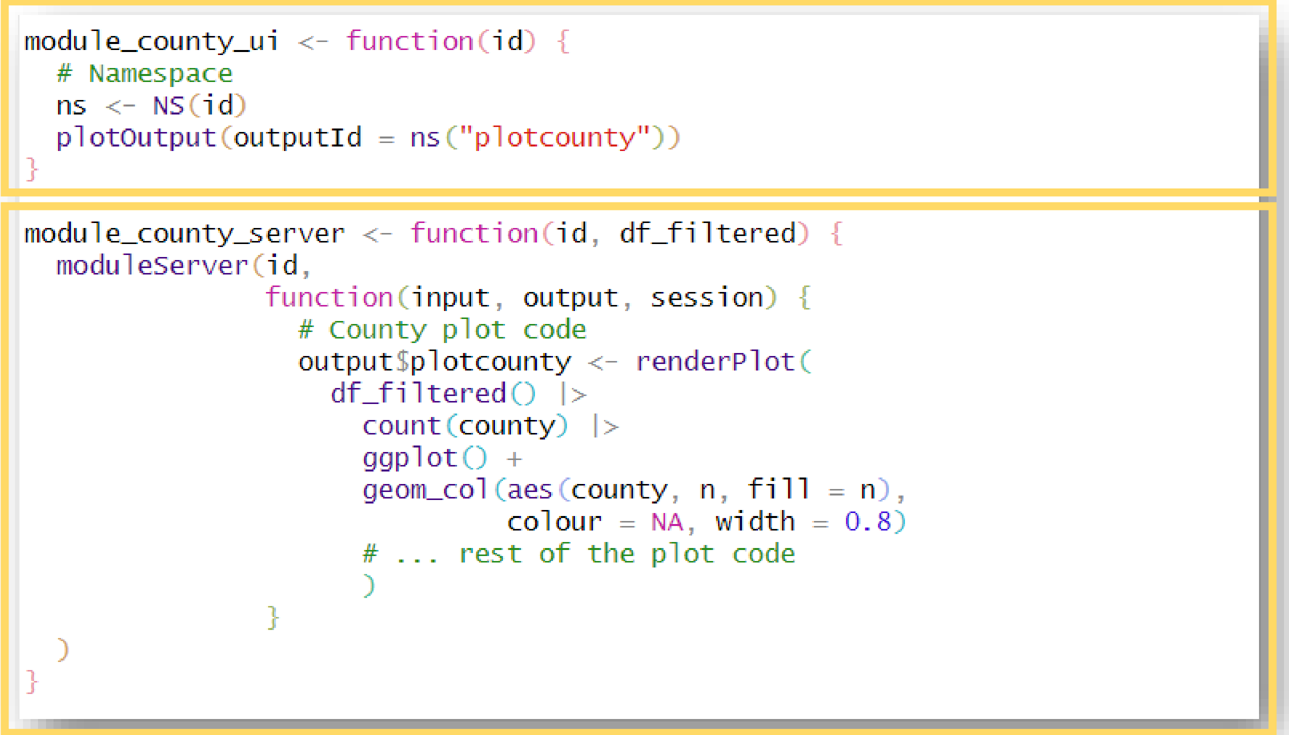
- Module 1: Module_input.R
  - Ui: Create state input ui
  - Server: Filter data based on state input
- Module 2: Module_countyplot.R
  - Ui: Create county plot ui
  - Server: Generate the county plot with ggplot code
- Module 3: Module_yearlyplot.R
  - Ui: Create yearly plot ui
  - Server: Generate the yearly plot with ggplot code

Original app.R file



```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Module 2: Module_countyplot.R

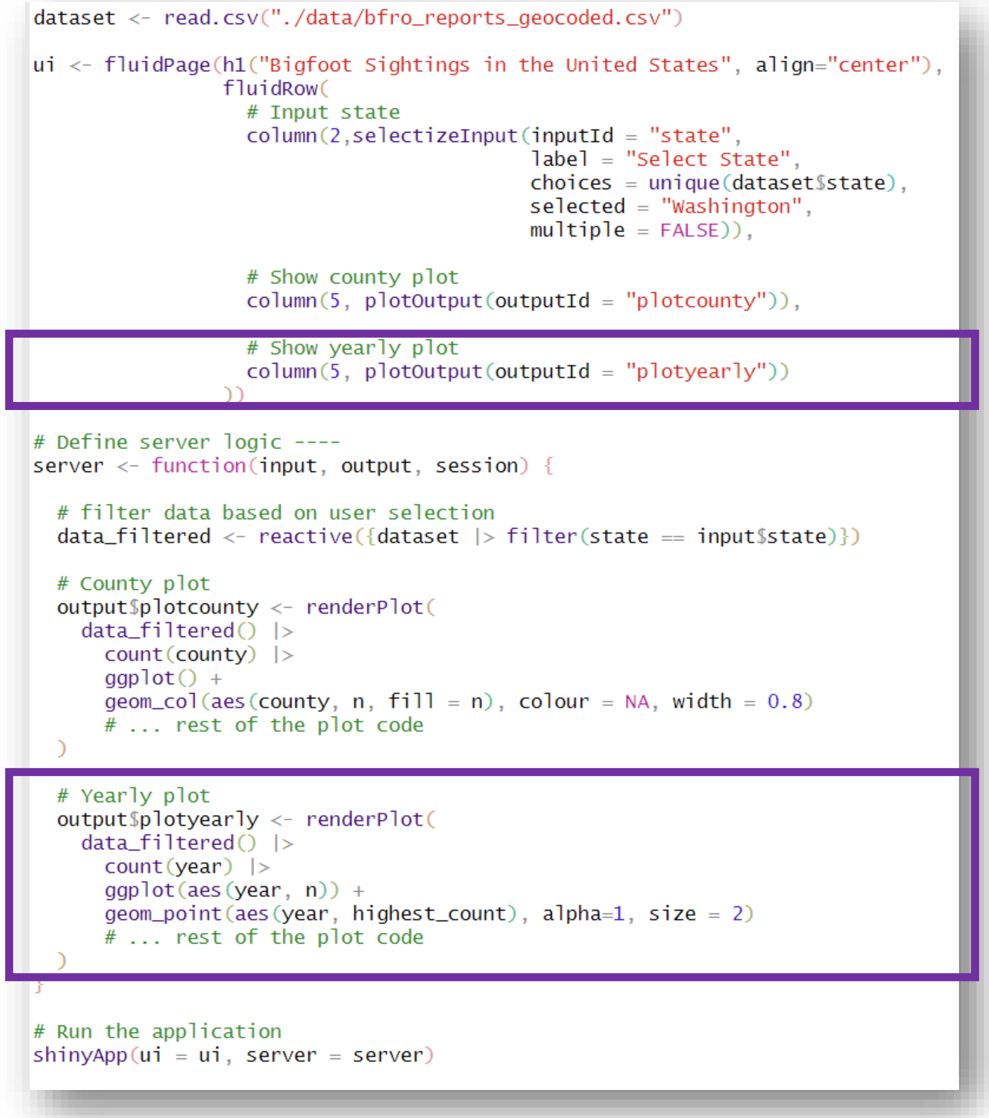


```r
module_county_ui <- function(id) {
  # Namespace
  ns <- NS(id)
  plotOutput(outputId = ns("plotcounty"))
}
```

```r
module_county_server <- function(id, df_filtered) {
  moduleServer(id,
               function(input, output, session) {
                 # County plot code
                 output$plotcounty <- renderPlot(
                   df_filtered() |>
                     count(county) |>
                     ggplot() +
                     geom_col(aes(county, n, fill = n),
                              colour = NA, width = 0.8)
                     # ... rest of the plot code
                     )
               }
  )
}
```

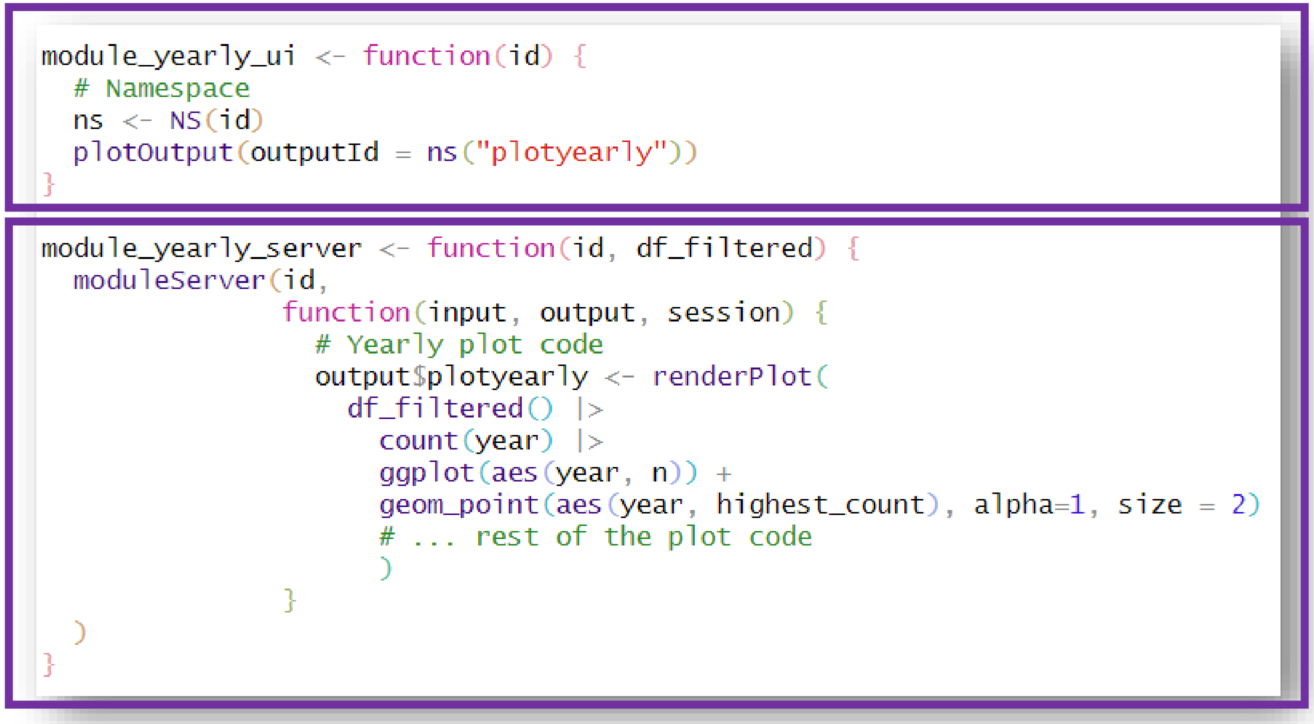
Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/

Original app.R file



```
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Module 3: Module_yearlyplot.R

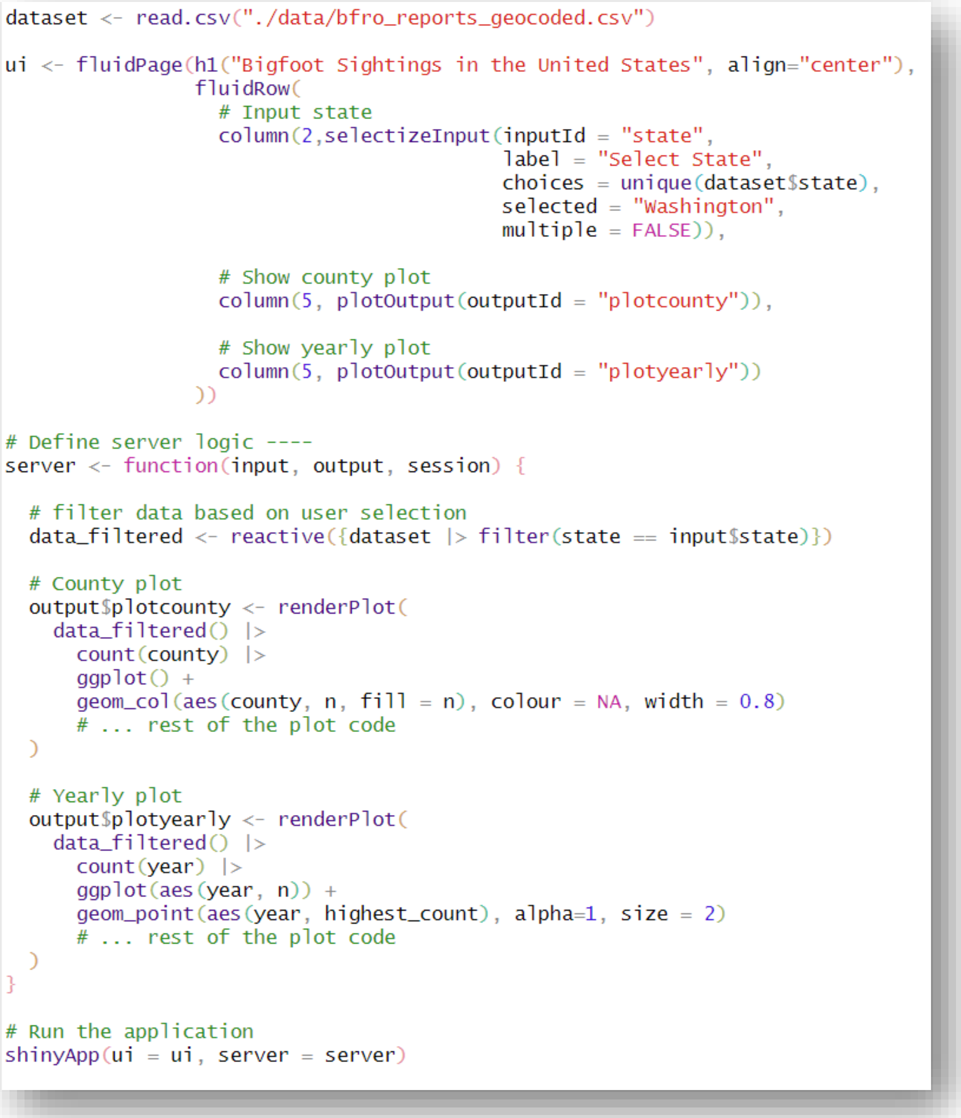


```
module_yearly_ui <- function(id) {
  # Namespace
  ns <- NS(id)
  plotOutput(outputId = ns("plotyearly"))
}
```

```
module_yearly_server <- function(id, df_filtered) {
  moduleServer(id,
               function(input, output, session) {
                 # Yearly plot code
                 output$plotyearly <- renderPlot(
                   df_filtered() |>
                     count(year) |>
                     ggplot(aes(year, n)) +
                     geom_point(aes(year, highest_count), alpha=1, size = 2)
                     # ... rest of the plot code
                     )
               }
  )
}
```

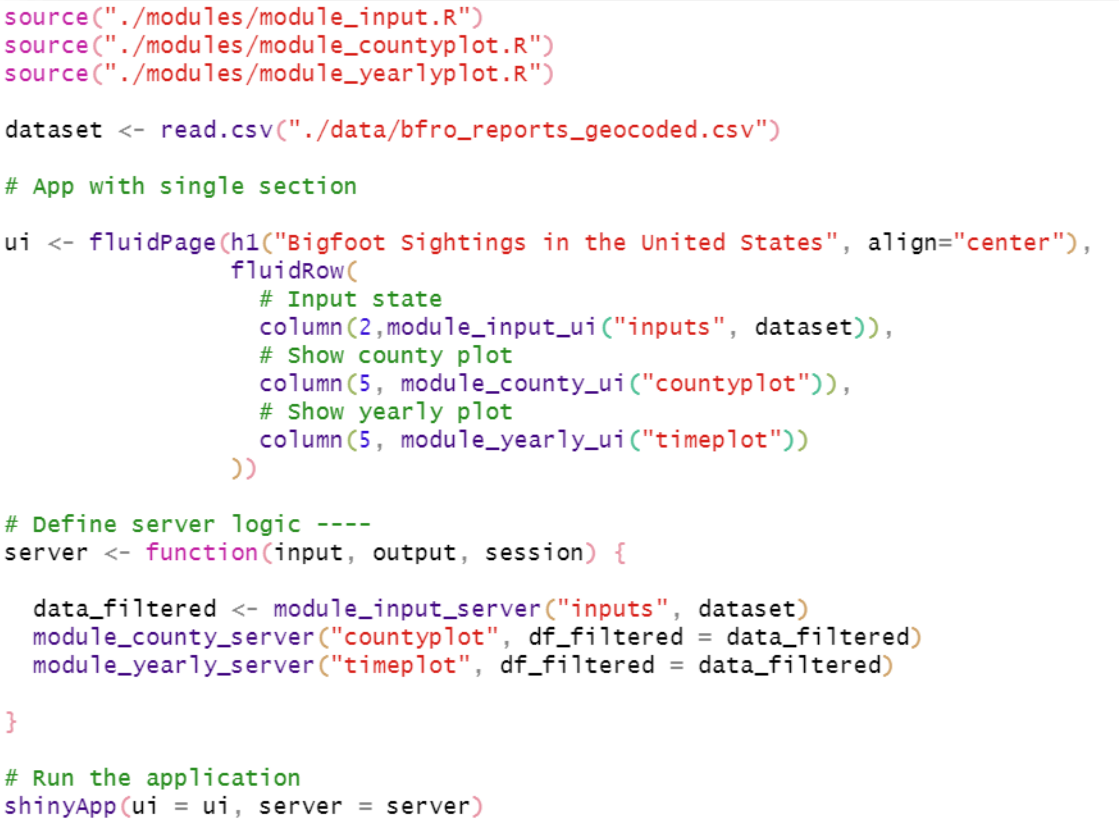
Step 5: Call modules from the main app.R file

Original app.R file



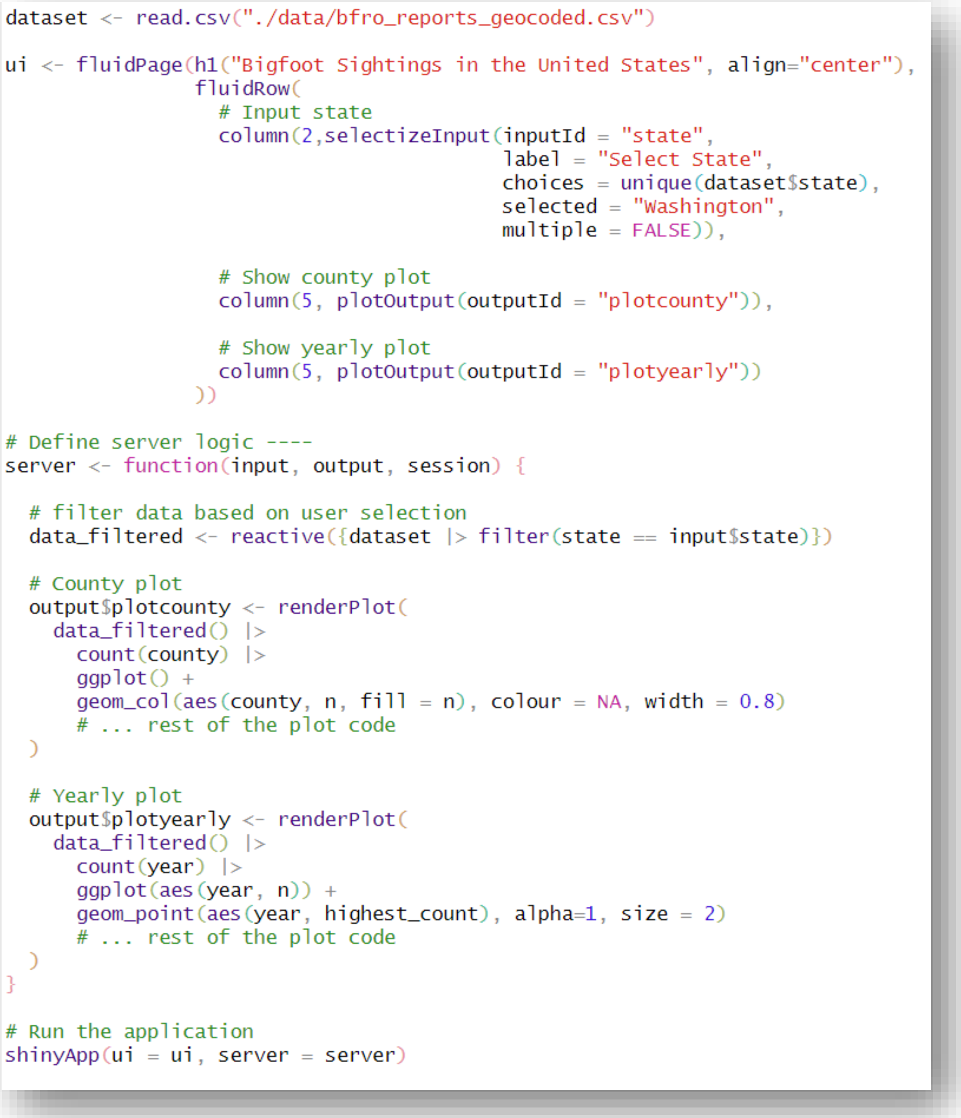
```
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Updated app.R file



```
source("./modules/module_input.R")
source("./modules/module_countyplot.R")
source("./modules/module_yearlyplot.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

# App with single section

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,module_input_ui("inputs", dataset)),
                  # Show county plot
                  column(5, module_county_ui("countyplot")),
                  # Show yearly plot
                  column(5, module_yearly_ui("timeplot"))
                ))

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

  data_filtered <- module_input_server("inputs", dataset)
  module_county_server("countyplot", df_filtered = data_filtered)
  module_yearly_server("timeplot", df_filtered = data_filtered)

}

# Run the application
shinyApp(ui = ui, server = server)
```

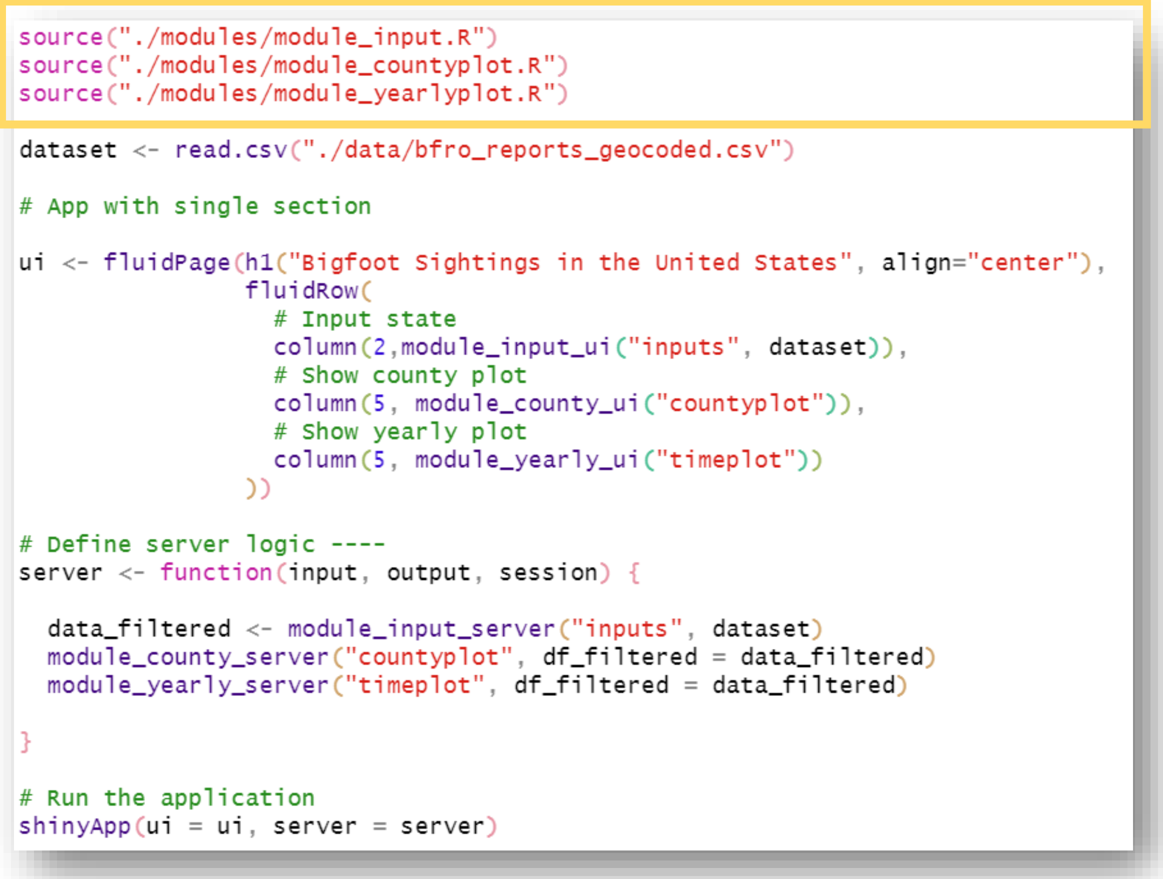
Original app.R file



```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

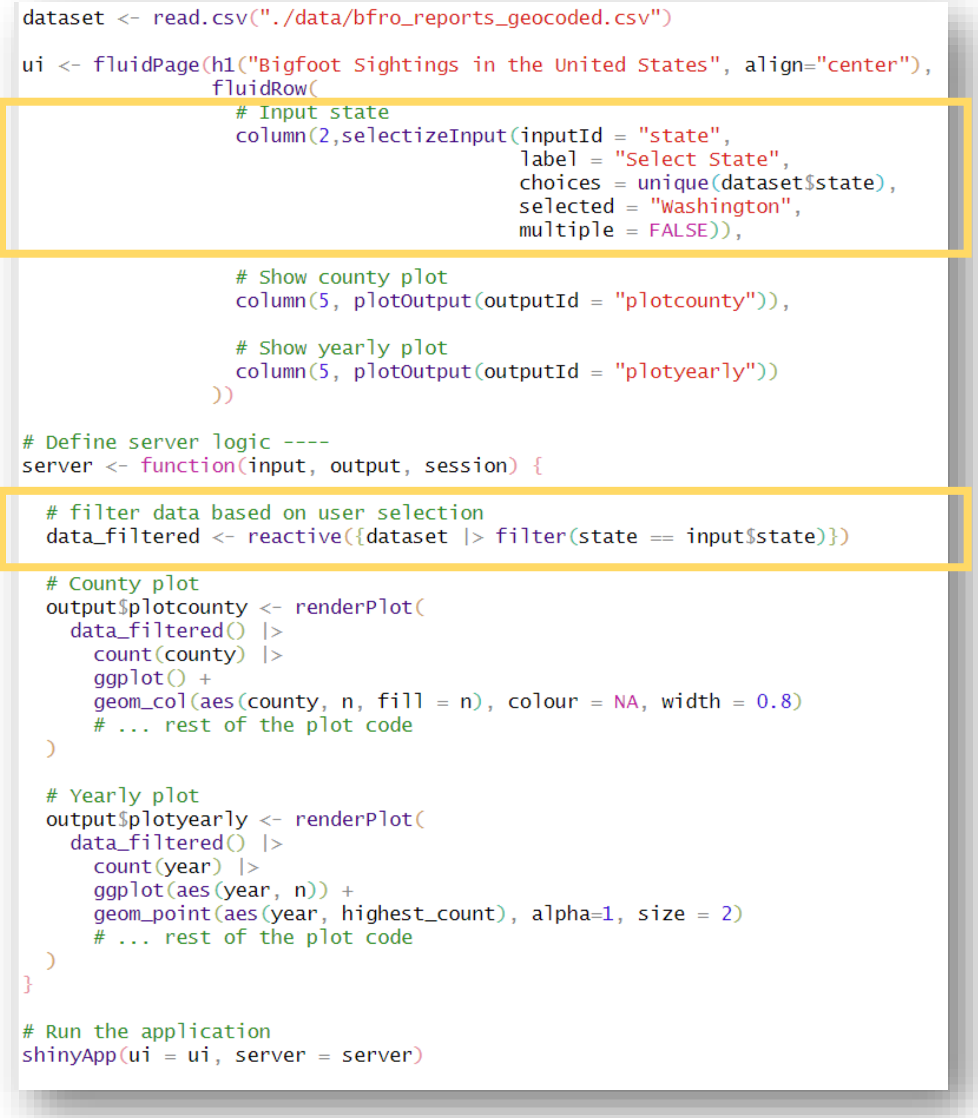
Source the module files

Updated app.R file



```r
source("./modules/module_input.R")
source("./modules/module_countyplot.R")
source("./modules/module_yearlyplot.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

# App with single section

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,module_input_ui("inputs", dataset)),
                  # Show county plot
                  column(5, module_county_ui("countyplot")),
                  # Show yearly plot
                  column(5, module_yearly_ui("timeplot"))
                ))

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

  data_filtered <- module_input_server("inputs", dataset)
  module_county_server("countyplot", df_filtered = data_filtered)
  module_yearly_server("timeplot", df_filtered = data_filtered)

}

# Run the application
shinyApp(ui = ui, server = server)
```

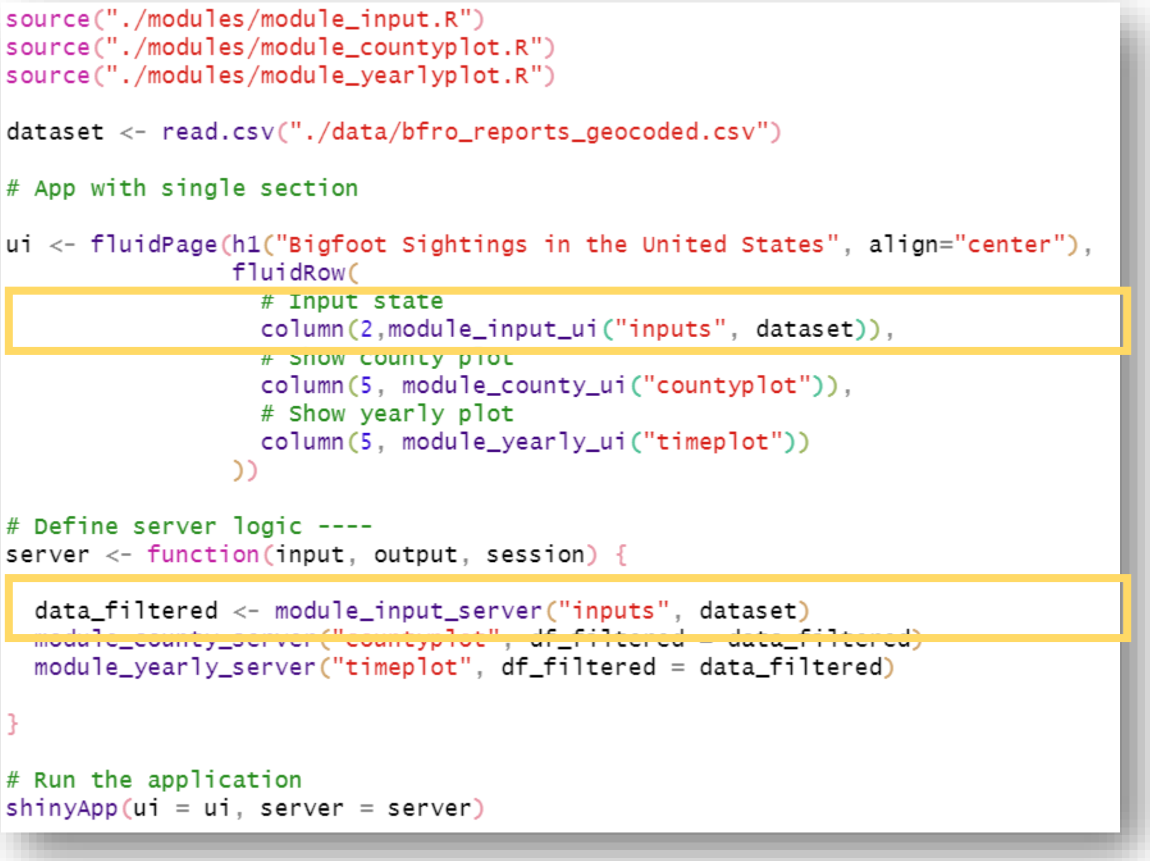
Original app.R file



```
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Call state input ui and server modules with “same” id

Updated app.R file

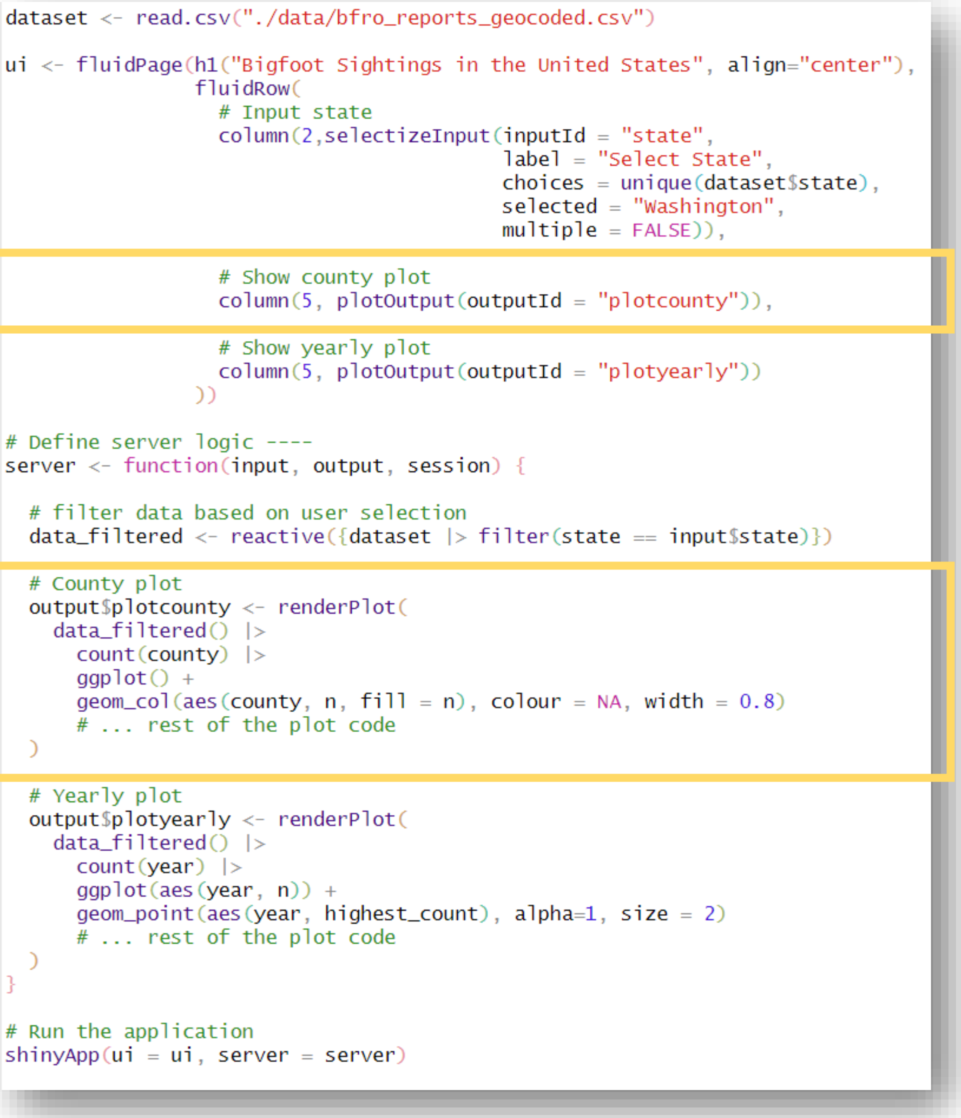


```
source("./modules/module_input.R")
source("./modules/module_countyplot.R")
source("./modules/module_yearlyplot.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

# App with single section

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,module_input_ui("inputs", dataset)),
                  # Show county plot
                  column(5, module_county_ui("countyplot")),
                  # Show yearly plot
                  column(5, module_yearly_ui("timeplot"))
                ))

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

  data_filtered <- module_input_server("inputs", dataset)
  module_county_server("countyplot", df_filtered = data_filtered)
  module_yearly_server("timeplot", df_filtered = data_filtered)

}

# Run the application
shinyApp(ui = ui, server = server)
```

Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/

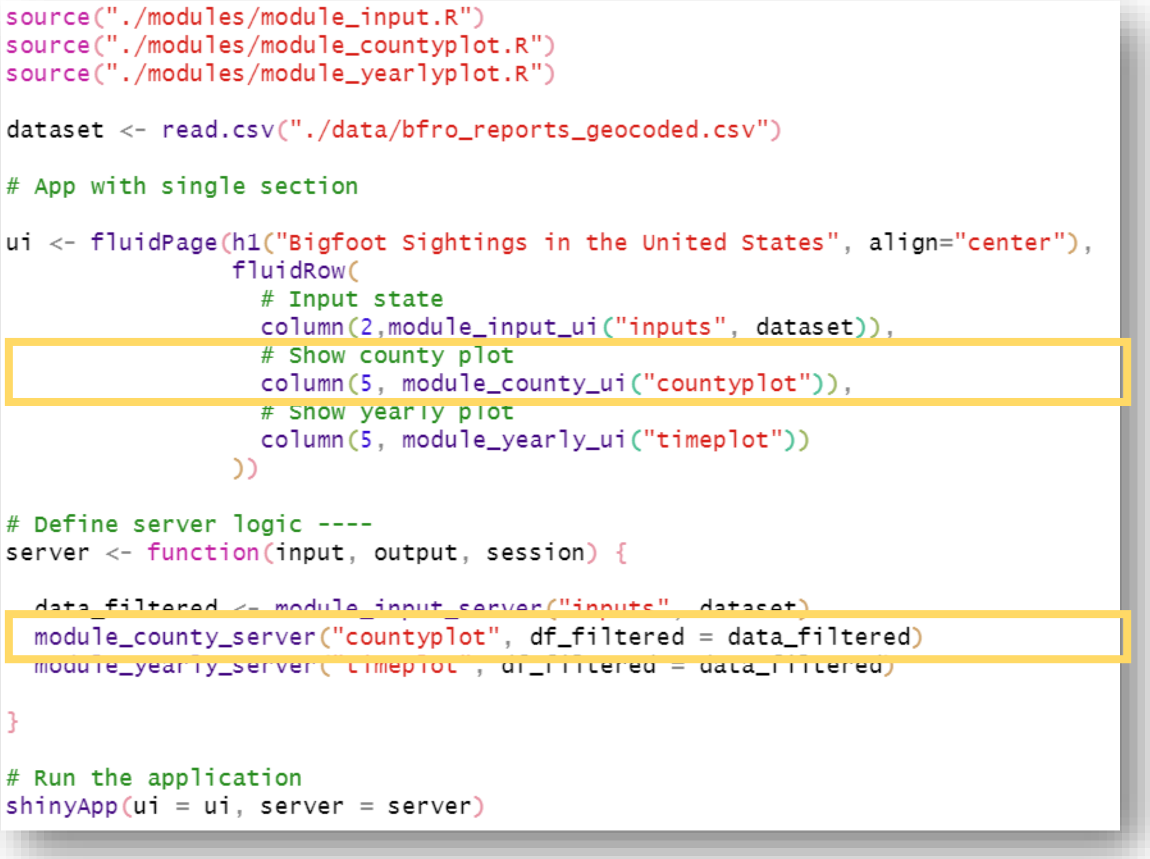
Similarly, call country plot ui and server modules

Original app.R file



```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

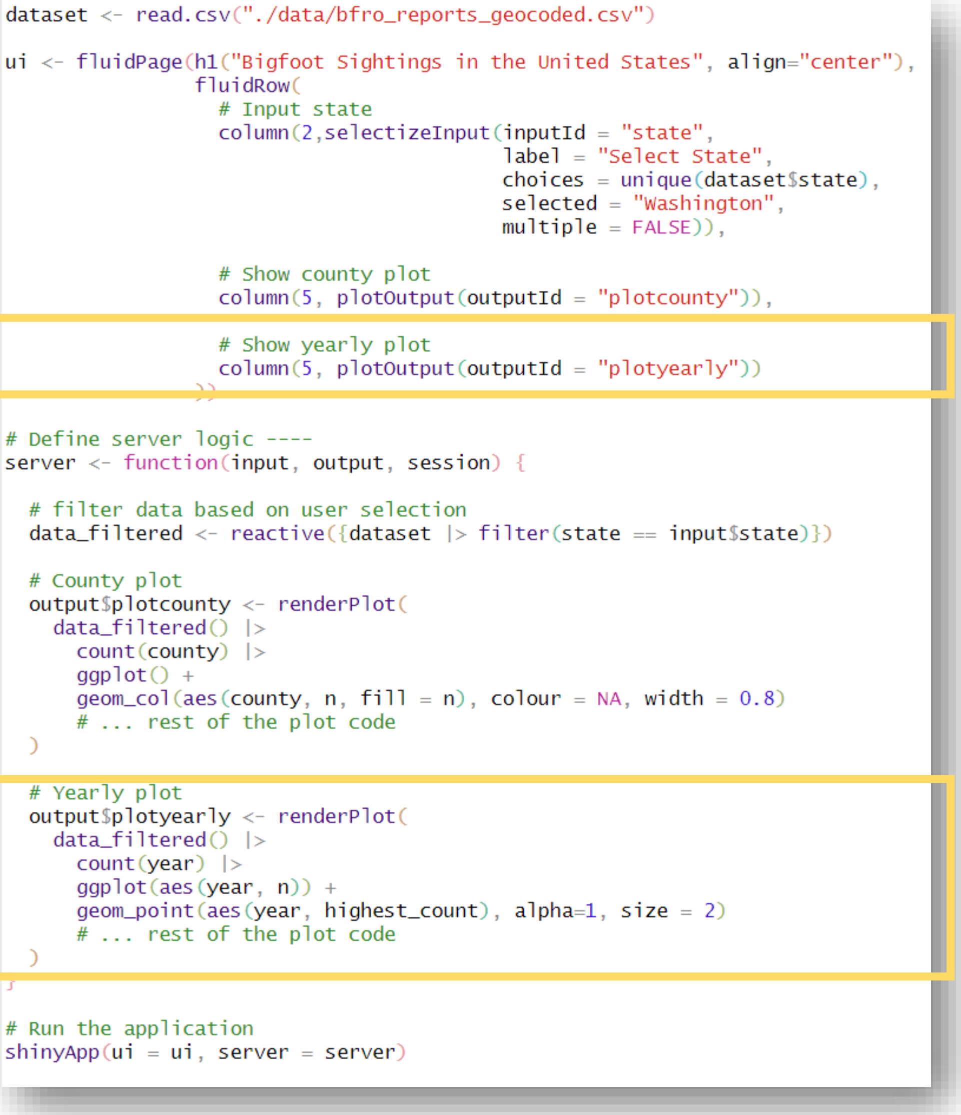
Updated app.R file



```r
source("./modules/module_input.R")
source("./modules/module_countyplot.R")
source("./modules/module_yearlyplot.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

# App with single section

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,module_input_ui("inputs", dataset)),
                  # Show county plot
                  column(5, module_county_ui("countyplot")),
                  # Show yearly plot
                  column(5, module_yearly_ui("timeplot"))
                ))

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

  data_filtered <- module_input_server("inputs", dataset)
  module_county_server("countyplot", df_filtered = data_filtered)
  module_yearly_server("timeplot", df_filtered = data_filtered)

}

# Run the application
shinyApp(ui = ui, server = server)
```

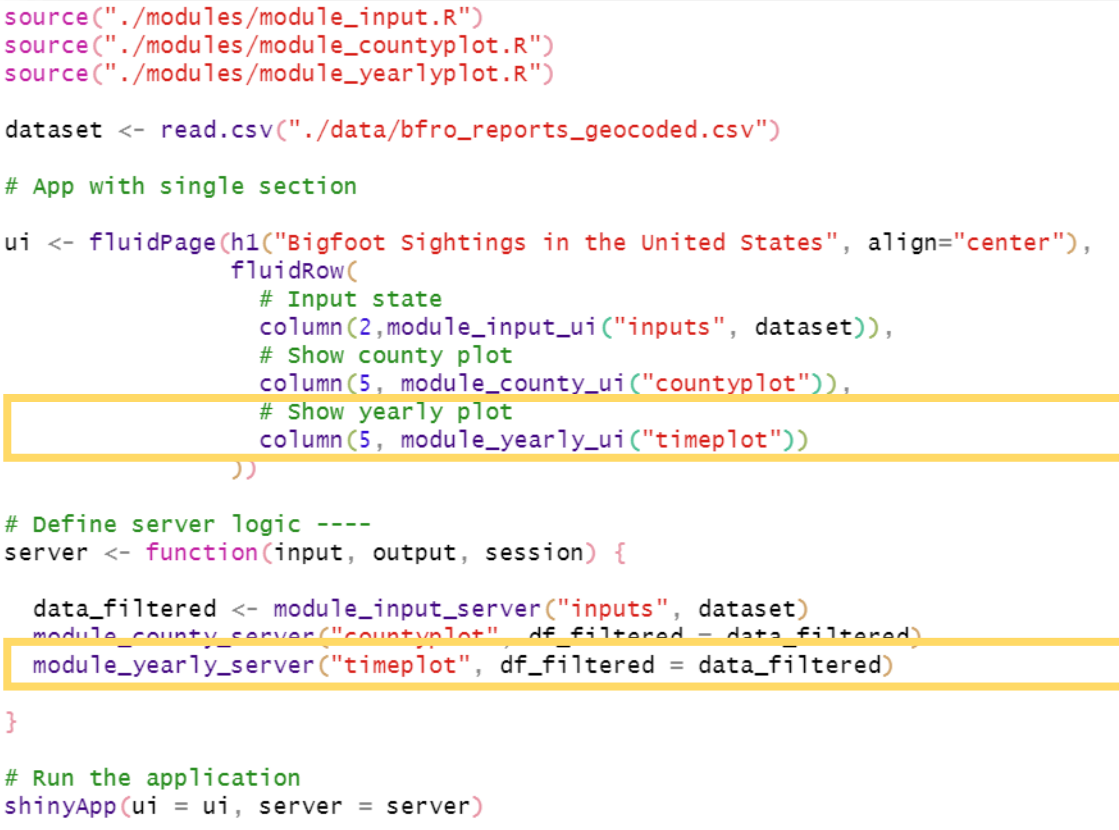
Similarly, call yearly plot ui and server modules

Original app.R file



```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

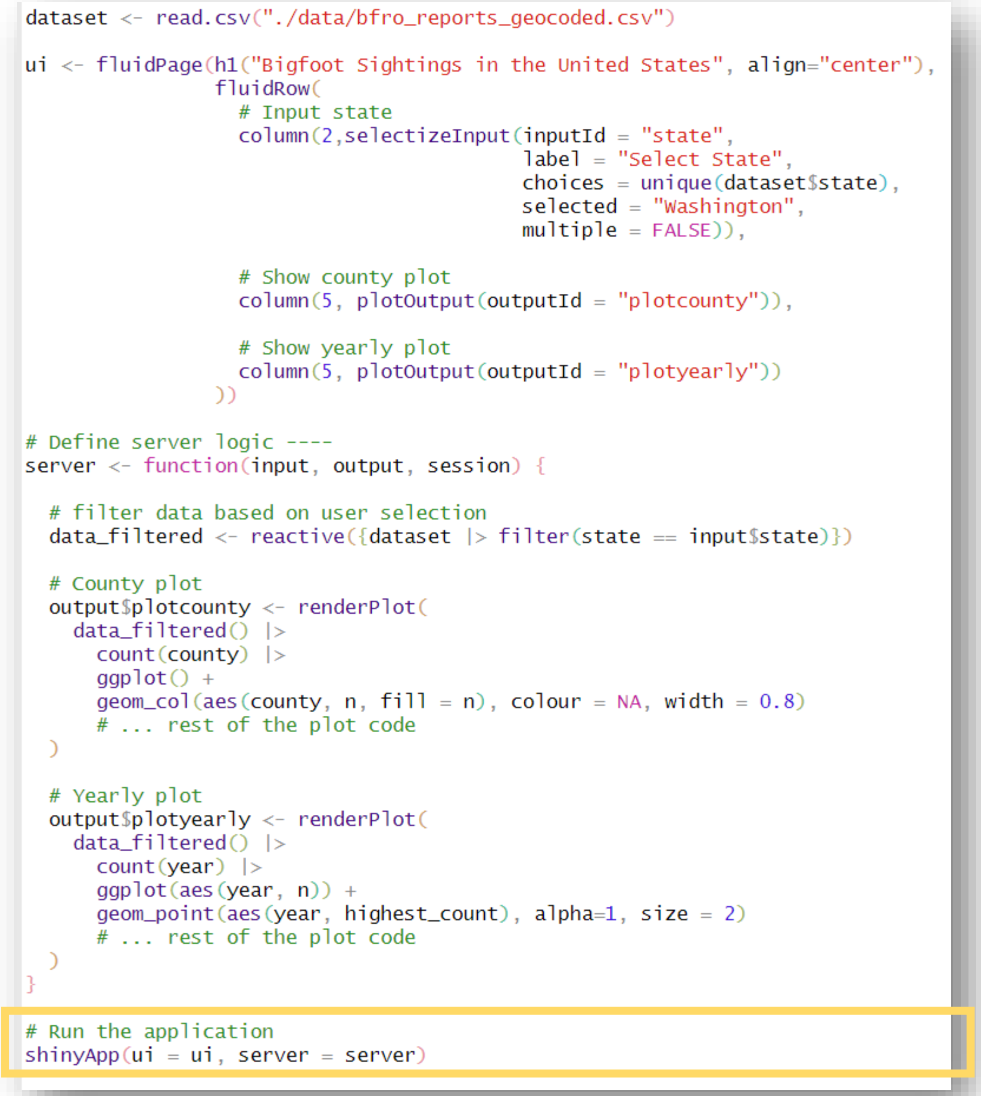
Updated app.R file



```r
source("./modules/module_input.R")
source("./modules/module_countyplot.R")
source("./modules/module_yearlyplot.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

# App with single section

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,module_input_ui("inputs", dataset)),
                  # Show county plot
                  column(5, module_county_ui("countyplot")),
                  # Show yearly plot
                  column(5, module_yearly_ui("timeplot"))
                ))

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

  data_filtered <- module_input_server("inputs", dataset)
  module_county_server("countyplot", df_filtered = data_filtered)
  module_yearly_server("timeplot", df_filtered = data_filtered)

}

# Run the application
shinyApp(ui = ui, server = server)
```

Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/

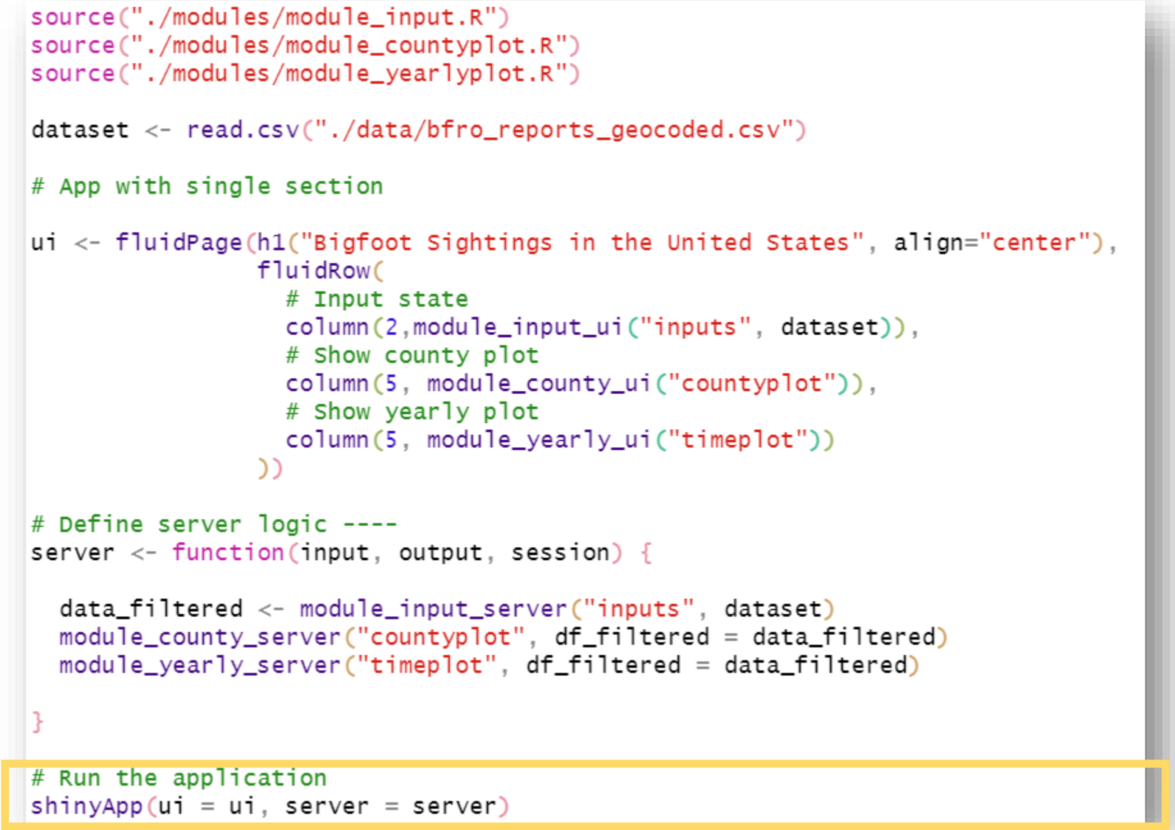
Original app.R file



```
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Run the shiny app

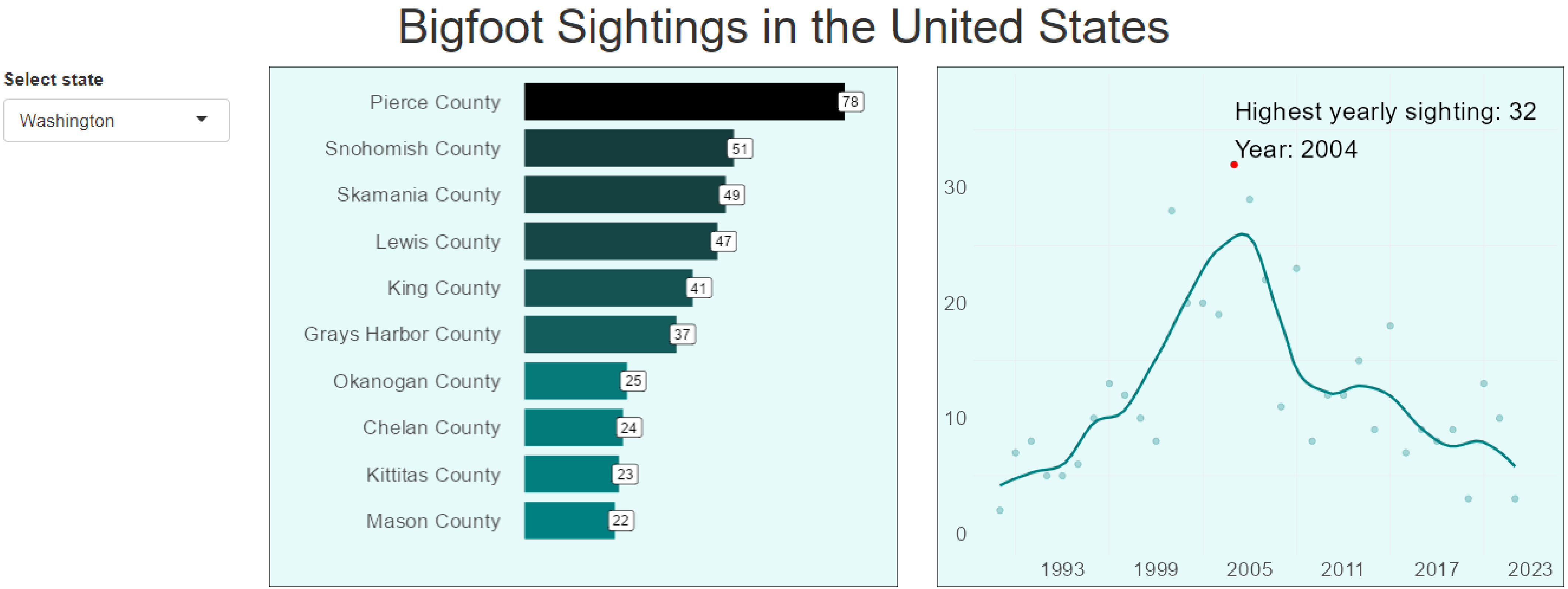
Updated app.R file



```
source("./modules/module_input.R")
source("./modules/module_countyplot.R")
source("./modules/module_yearlyplot.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

# App with single section

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,module_input_ui("inputs", dataset)),
                  # Show county plot
                  column(5, module_county_ui("countyplot")),
                  # Show yearly plot
                  column(5, module_yearly_ui("timeplot"))
                ))

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

  data_filtered <- module_input_server("inputs", dataset)
  module_county_server("countyplot", df_filtered = data_filtered)
  module_yearly_server("timeplot", df_filtered = data_filtered)

}
# Run the application
shinyApp(ui = ui, server = server)
```

app.R is so clean and simple to read!!!

Shiny output



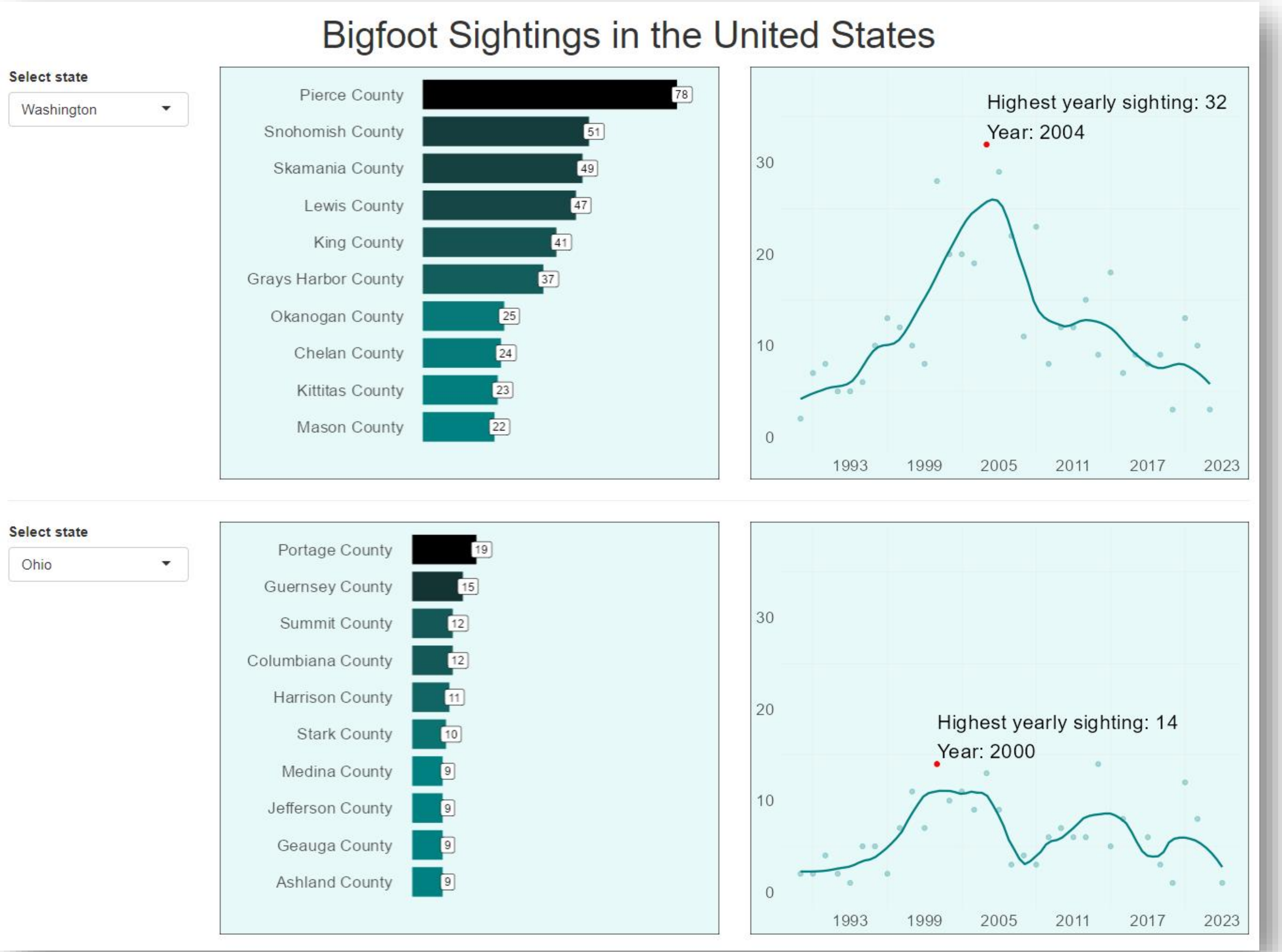
Bigfoot Sightings in the United States

Select state

Washington

Now, let’s replicate input and output plots for two state comparison

Like this ->



Bigfoot Sightings in the United States

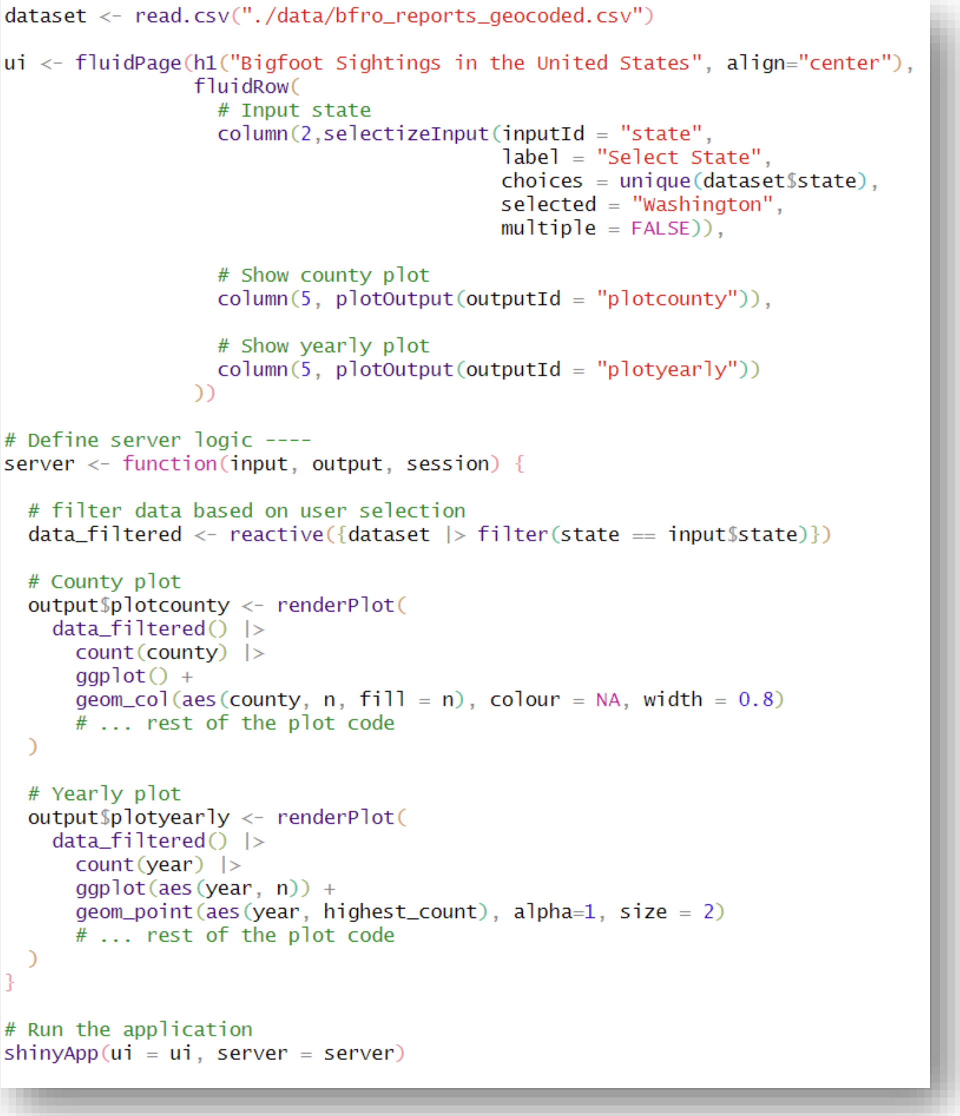
Select state

Washington

Select state

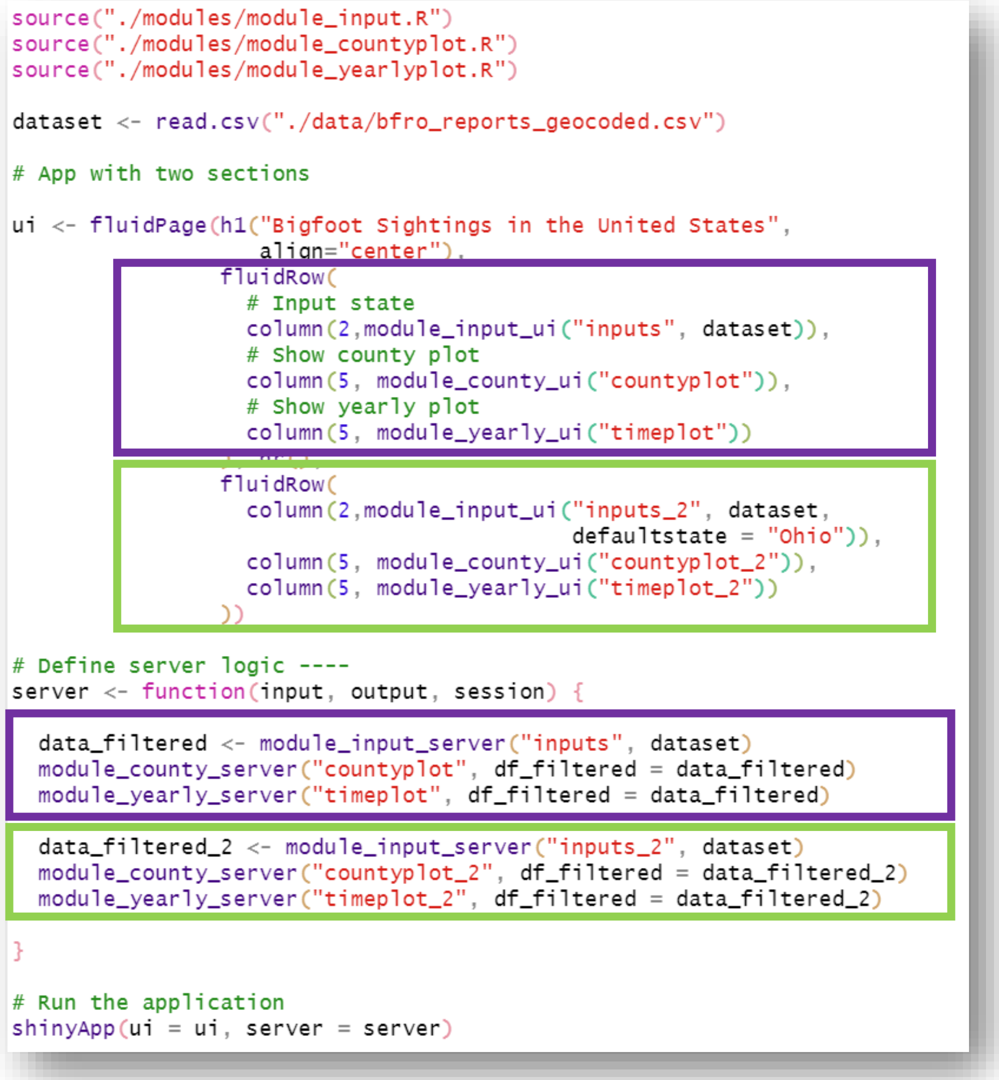
Ohio

Original app.R file



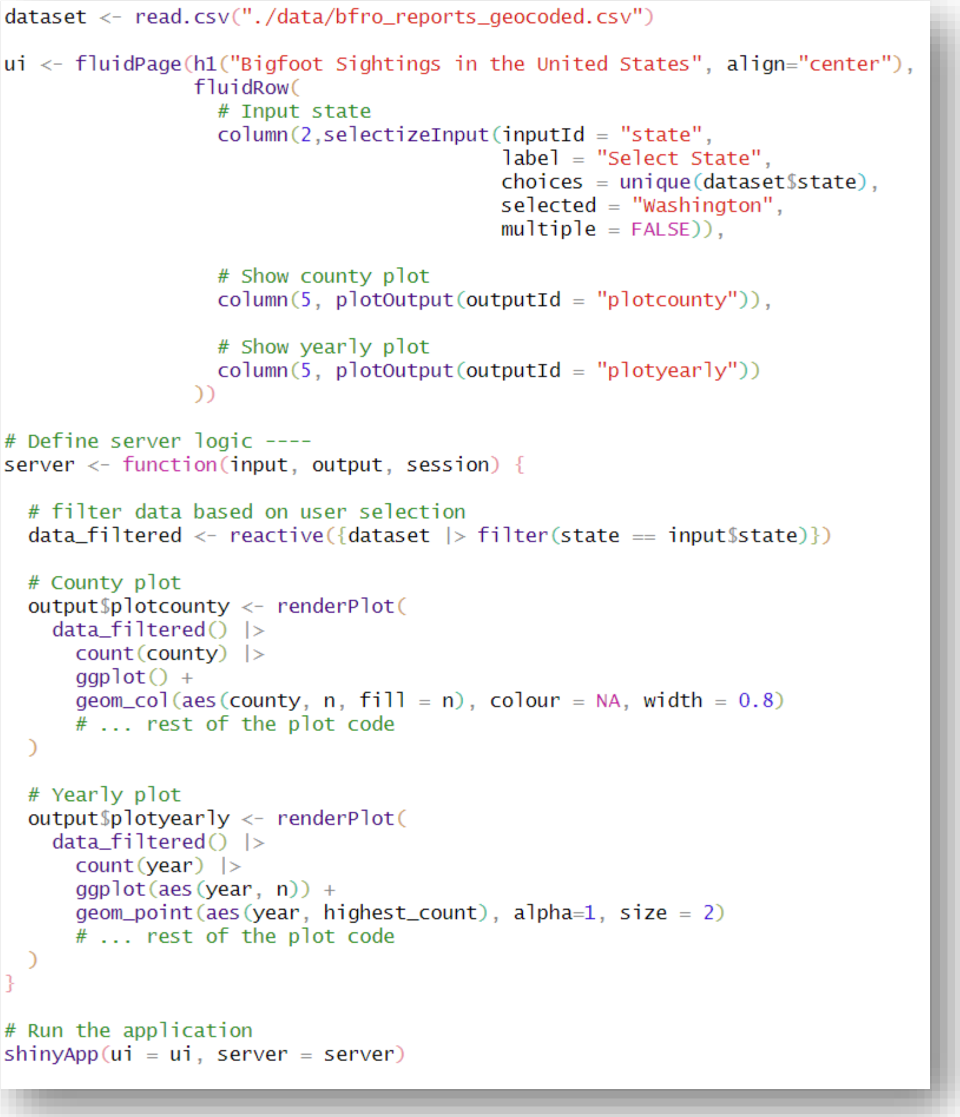
```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

Updated app.R file – single module, multiple calls



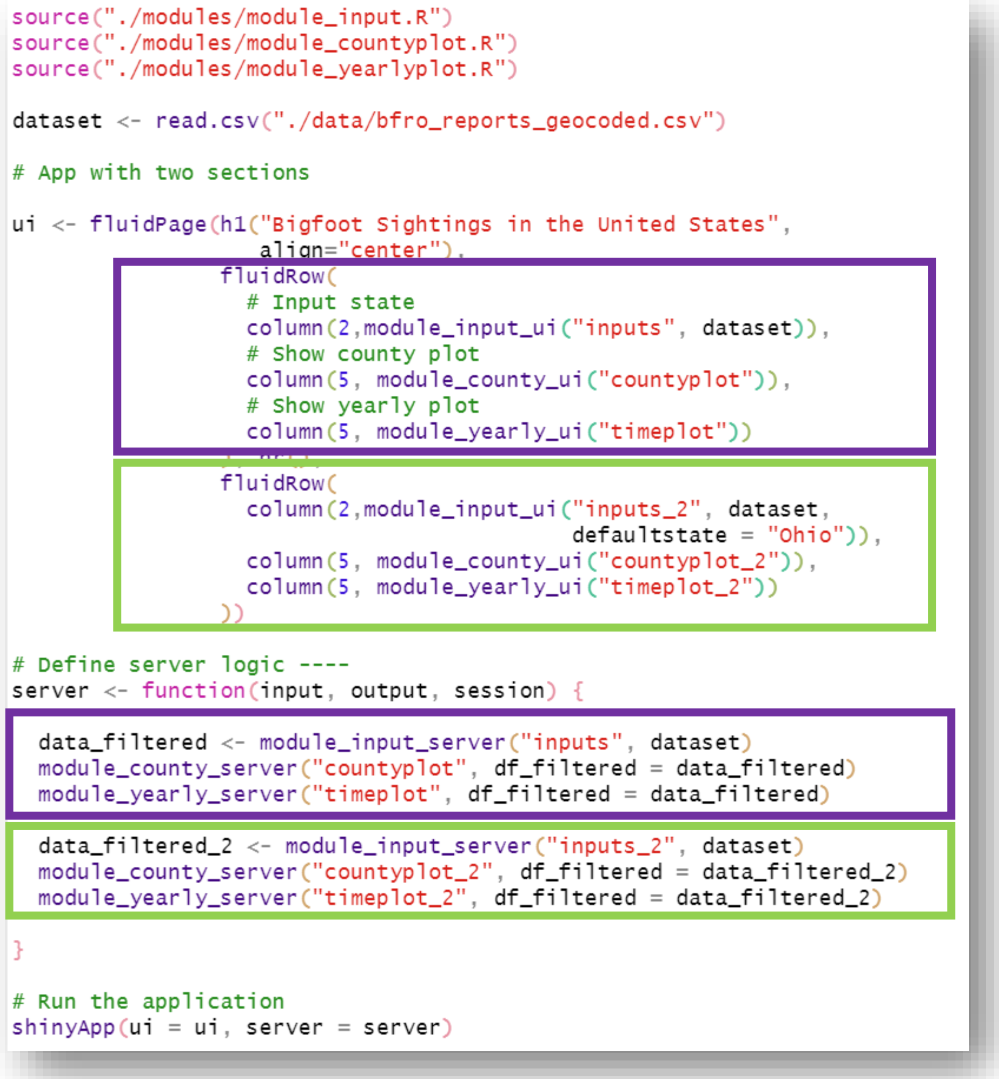
```r
source("./modules/module_input.R")
source("./modules/module_countyplot.R")
source("./modules/module_yearlyplot.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

# App with two sections

ui <- fluidPage(h1("Bigfoot Sightings in the United States",
                   align="center"),
                fluidRow(
                  # Input state
                  column(2,module_input_ui("inputs", dataset)),
                  # Show county plot
                  column(5, module_county_ui("countyplot")),
                  # Show yearly plot
                  column(5, module_yearly_ui("timeplot"))
                ), br(),
                fluidRow(
                  column(2,module_input_ui("inputs_2", dataset,
                                           defaultstate = "Ohio")),
                  column(5, module_county_ui("countyplot_2")),
                  column(5, module_yearly_ui("timeplot_2"))
                ))

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

  data_filtered <- module_input_server("inputs", dataset)
  module_county_server("countyplot", df_filtered = data_filtered)
  module_yearly_server("timeplot", df_filtered = data_filtered)

  data_filtered_2 <- module_input_server("inputs_2", dataset)
  module_county_server("countyplot_2", df_filtered = data_filtered_2)
  module_yearly_server("timeplot_2", df_filtered = data_filtered_2)

}

# Run the application
shinyApp(ui = ui, server = server)
```

Original app.R file



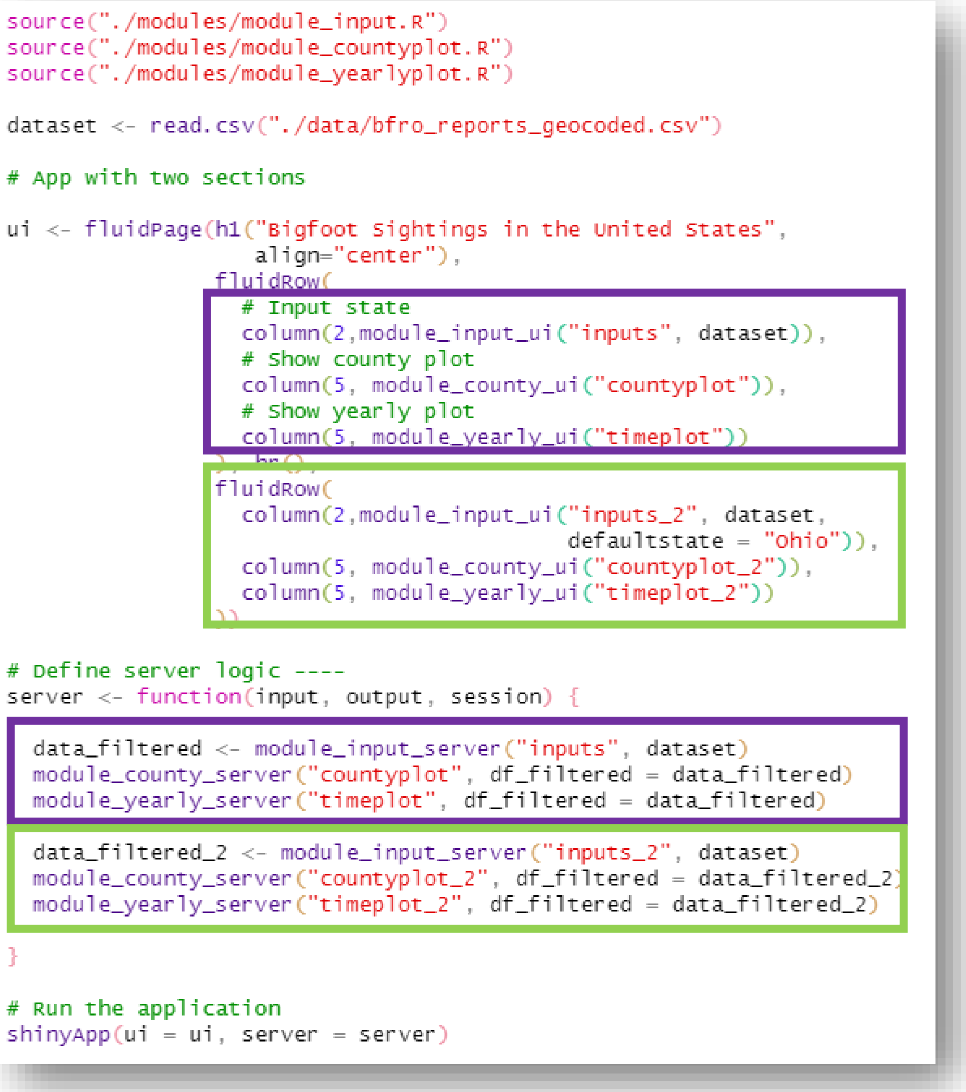
```r
dataset <- read.csv("./data/bfro_reports_geocoded.csv")

ui <- fluidPage(h1("Bigfoot Sightings in the United States", align="center"),
                fluidRow(
                  # Input state
                  column(2,selectizeInput(inputId = "state",
                                          label = "Select State",
                                          choices = unique(dataset$state),
                                          selected = "Washington",
                                          multiple = FALSE)),

                  # Show county plot
                  column(5, plotOutput(outputId = "plotcounty")),

                  # Show yearly plot
                  column(5, plotOutput(outputId = "plotyearly"))
                ))

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

  # filter data based on user selection
  data_filtered <- reactive({dataset |> filter(state == input$state)})

  # County plot
  output$plotcounty <- renderPlot(
    data_filtered() |>
      count(county) |>
      ggplot() +
      geom_col(aes(county, n, fill = n), colour = NA, width = 0.8)
      # ... rest of the plot code
  )

  # Yearly plot
  output$plotyearly <- renderPlot(
    data_filtered() |>
      count(year) |>
      ggplot(aes(year, n)) +
      geom_point(aes(year, highest_count), alpha=1, size = 2)
      # ... rest of the plot code
  )
}

# Run the application
shinyApp(ui = ui, server = server)
```

NOTE: Unique ids amongst each module calls



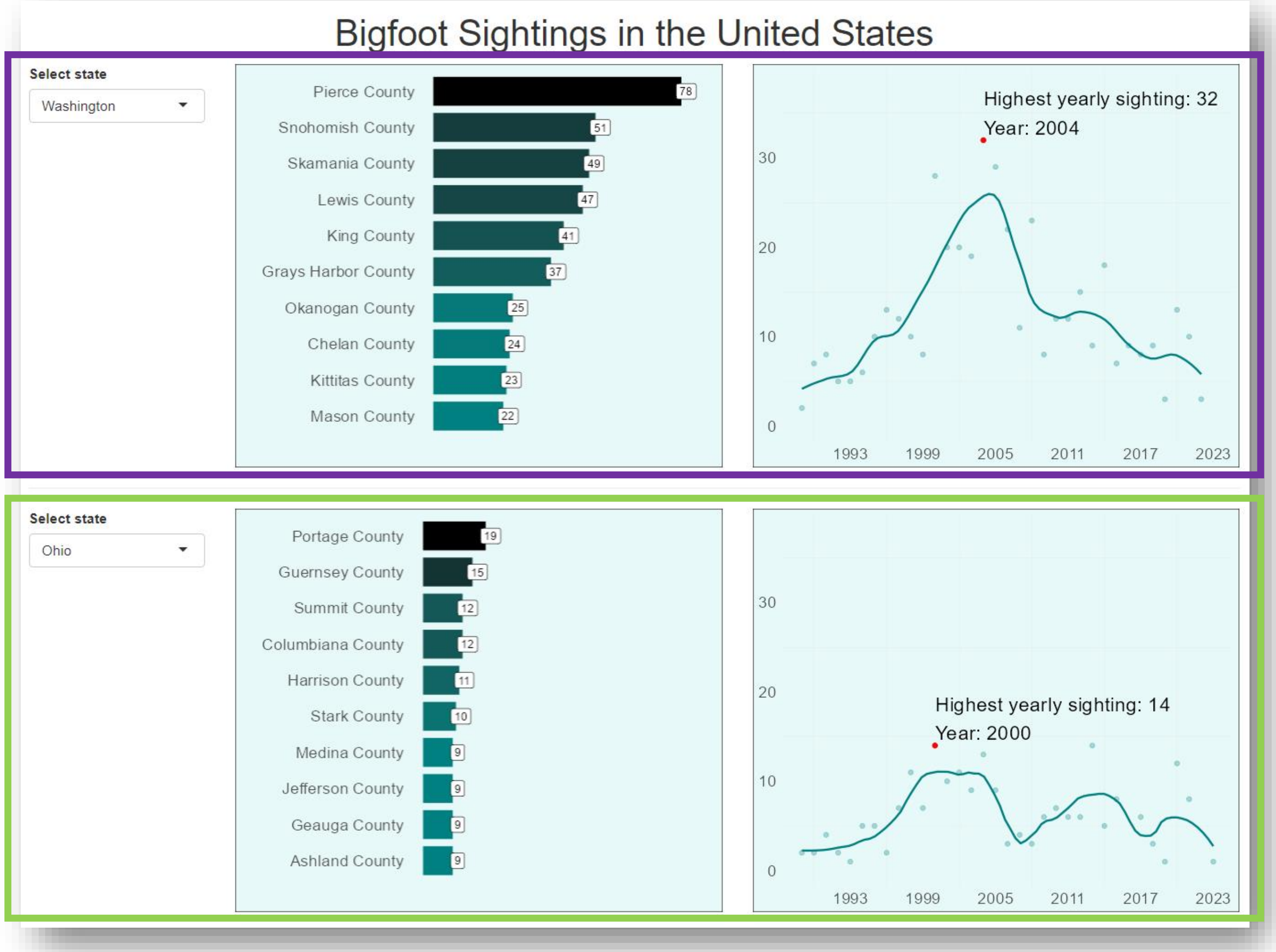
```r
source("./modules/module_input.R")
source("./modules/module_countyplot.R")
source("./modules/module_yearlyplot.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

# App with two sections

ui <- fluidPage(h1("Bigfoot Sightings in the United States",
                   align="center"),
                fluidRow(
                  # Input state
                  column(2,module_input_ui("inputs", dataset)),
                  # Show county plot
                  column(5, module_county_ui("countyplot")),
                  # Show yearly plot
                  column(5, module_yearly_ui("timeplot"))
                ), br(),
                fluidRow(
                  column(2,module_input_ui("inputs_2", dataset,
                                           defaultstate = "Ohio")),
                  column(5, module_county_ui("countyplot_2")),
                  column(5, module_yearly_ui("timeplot_2"))
                ))

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

  data_filtered <- module_input_server("inputs", dataset)
  module_county_server("countyplot", df_filtered = data_filtered)
  module_yearly_server("timeplot", df_filtered = data_filtered)

  data_filtered_2 <- module_input_server("inputs_2", dataset)
  module_county_server("countyplot_2", df_filtered = data_filtered_2)
  module_yearly_server("timeplot_2", df_filtered = data_filtered_2)

}

# Run the application
shinyApp(ui = ui, server = server)
```

Updated app.R file – single module, multiple calls



```
source("./modules/module_input.R")
source("./modules/module_countyplot.R")
source("./modules/module_yearlyplot.R")

dataset <- read.csv("./data/bfro_reports_geocoded.csv")

# App with two sections

ui <- fluidPage(h1("Bigfoot Sightings in the United States",
                   align="center"),
                fluidRow(
                  # Input state
                  column(2,module_input_ui("inputs", dataset)),
                  # Show county plot
                  column(5, module_county_ui("countyplot")),
                  # Show yearly plot
                  column(5, module_yearly_ui("timeplot"))
                ), hr(),
                fluidRow(
                  column(2,module_input_ui("inputs_2", dataset,
                                           defaultstate = "Ohio")),
                  column(5, module_county_ui("countyplot_2")),
                  column(5, module_yearly_ui("timeplot_2"))
                ))

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

  data_filtered <- module_input_server("inputs", dataset)
  module_county_server("countyplot", df_filtered = data_filtered)
  module_yearly_server("timeplot", df_filtered = data_filtered)

  data_filtered_2 <- module_input_server("inputs_2", dataset)
  module_county_server("countyplot_2", df_filtered = data_filtered_2)
  module_yearly_server("timeplot_2", df_filtered = data_filtered_2)

}

# Run the application
shinyApp(ui = ui, server = server)
```

Shiny App output



Bigfoot Sightings in the United States

Select state: Washington

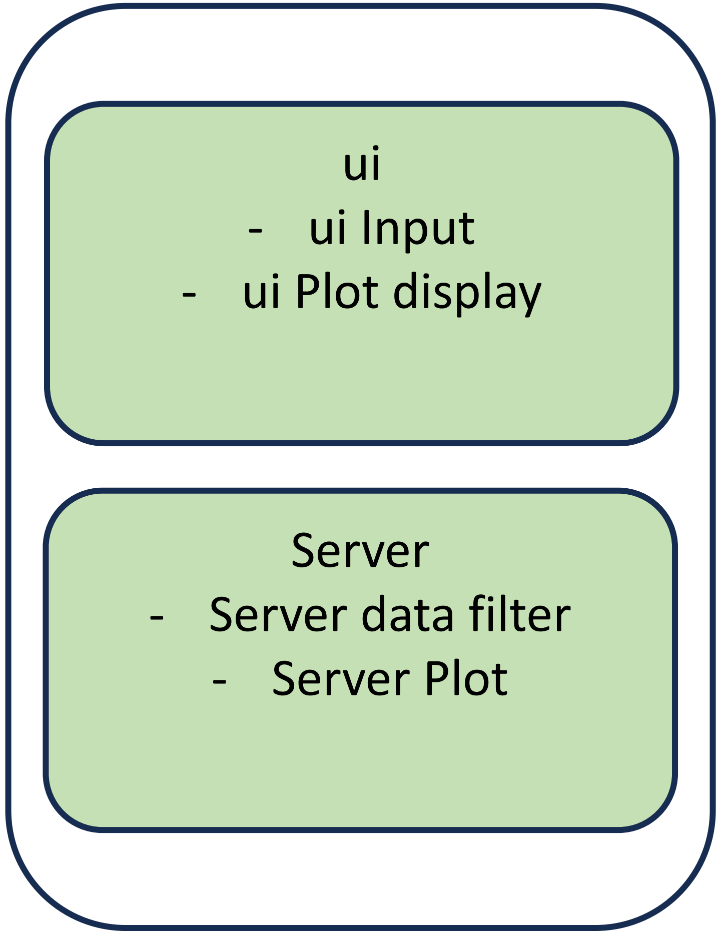
Highest yearly sighting: 32
Year: 2004

Select state: Ohio

Highest yearly sighting: 14
Year: 2000

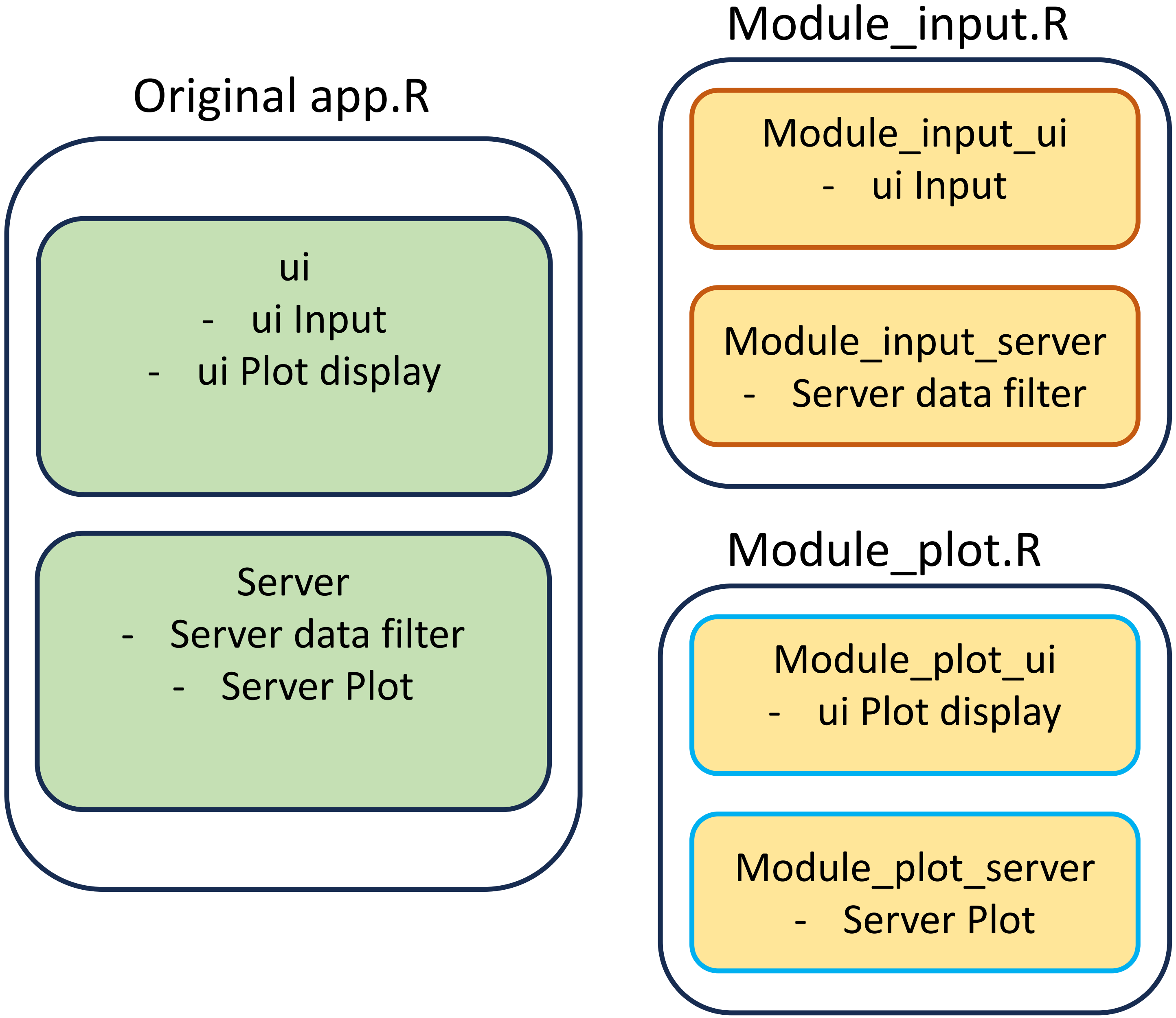
Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/

Original app.R



ui
- ui Input
- ui Plot display

Server
- Server data filter
- Server Plot



Original app.R

ui
- ui Input
- ui Plot display

Server
- Server data filter
- Server Plot

Module_input.R

Module_input_ui
- ui Input

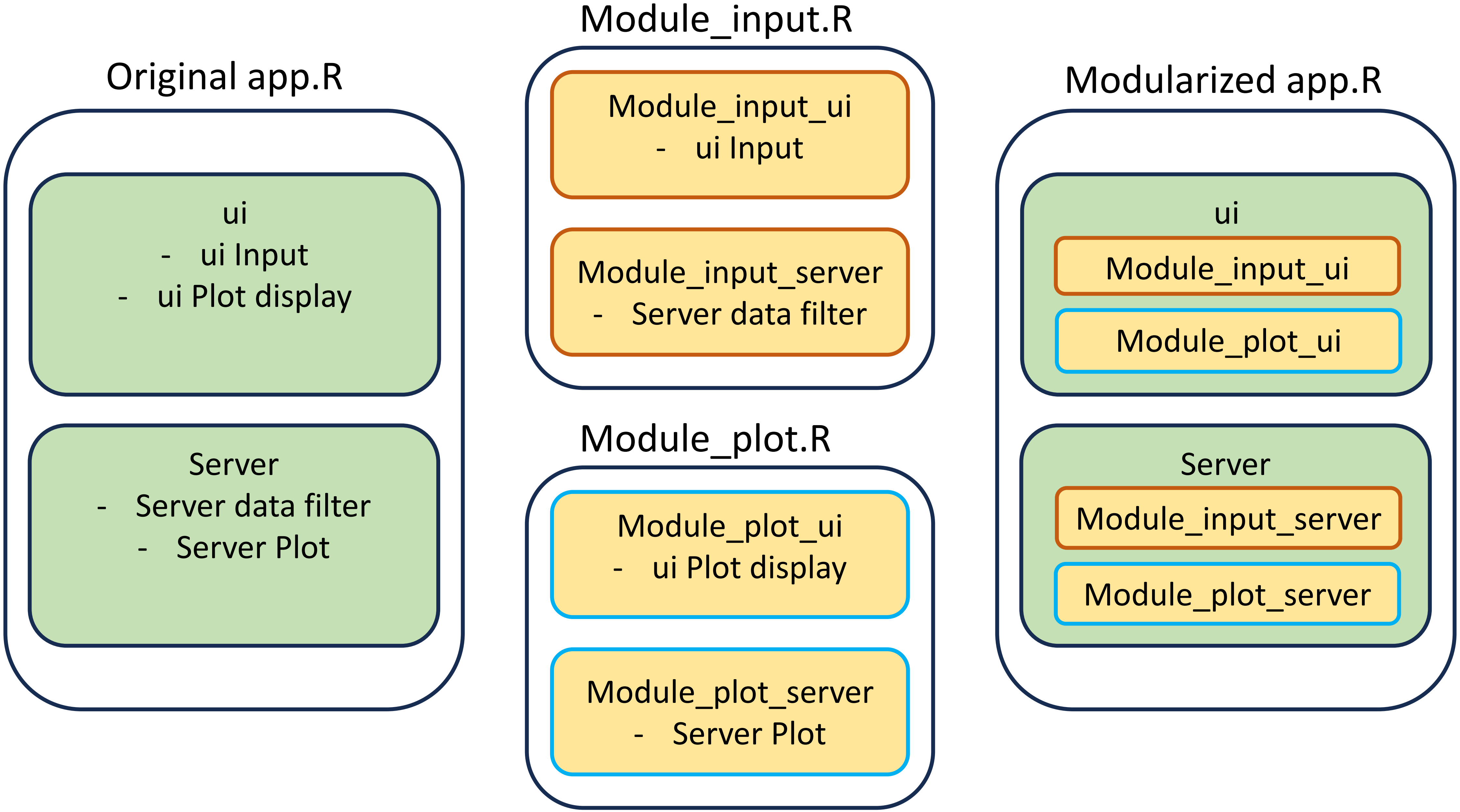
Module_input_server
- Server data filter

Module_plot.R

Module_plot_ui
- ui Plot display

Module_plot_server
- Server Plot

Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/



Original app.R

ui
- ui Input
- ui Plot display

Server
- Server data filter
- Server Plot

Module_input.R

Module_input_ui
- ui Input

Module_input_server
- Server data filter

Module_plot.R

Module_plot_ui
- ui Plot display

Module_plot_server
- Server Plot

Modularized app.R

ui

Module_input_ui

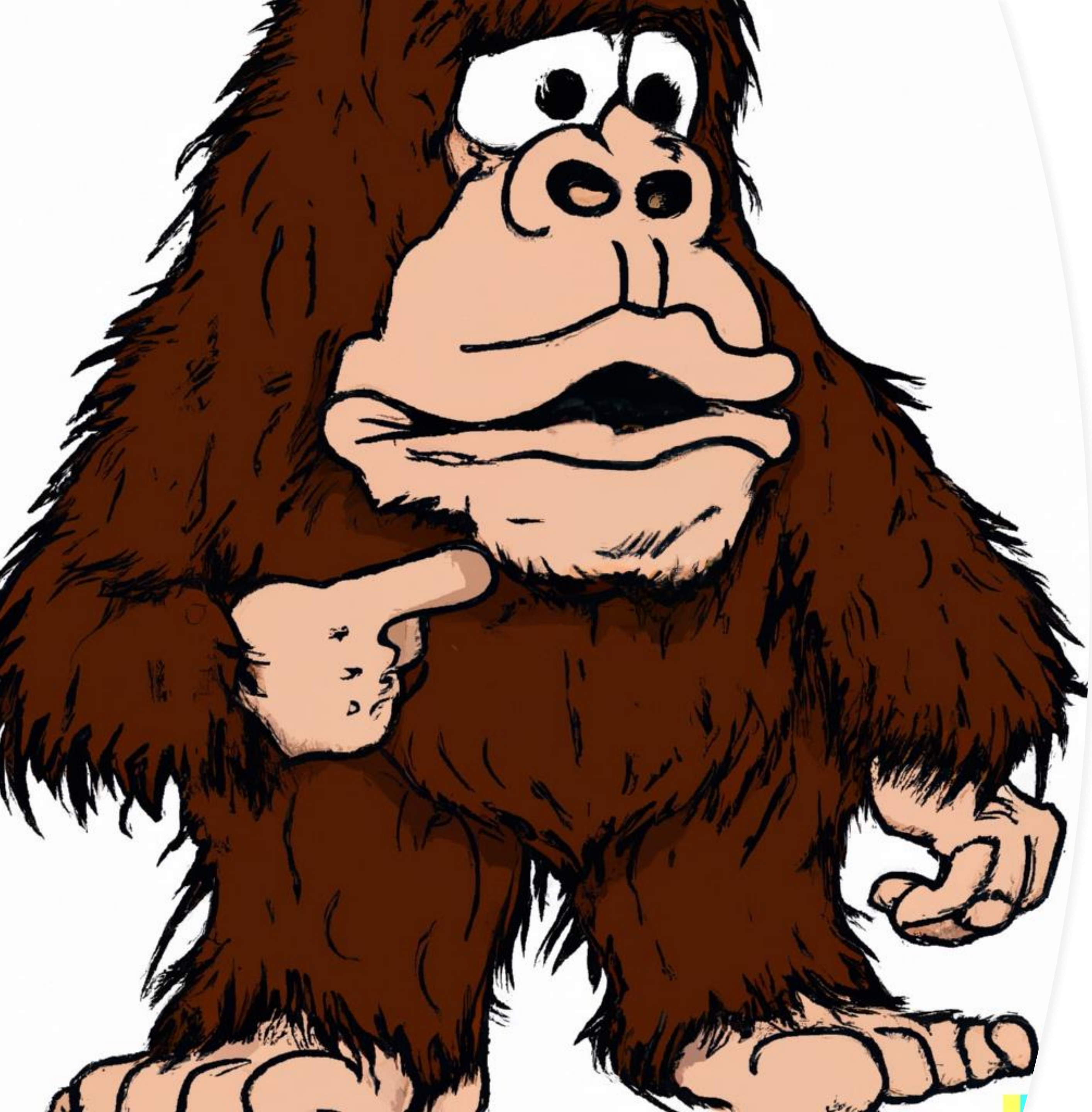
Module_plot_ui

Server

Module_input_server

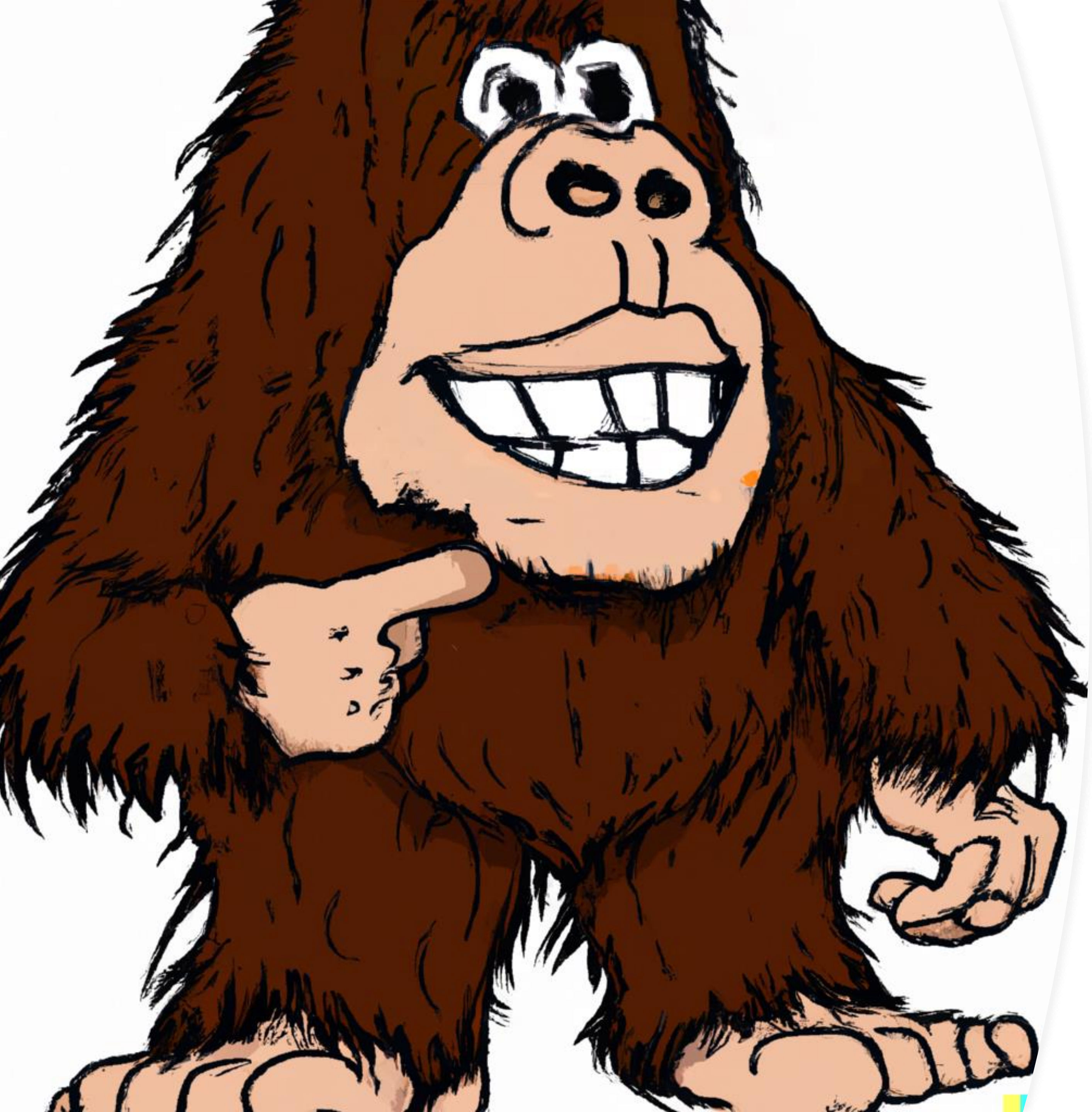
Module_plot_server

Deepsha Menghani
https://deepshamenghani.quarto.pub/dmenghani/



The challenges with a monolith app.R file

- Difficult to keep track of components
- Challenging to debug if something fails
- Harder for someone new to learn and contribute
- Tedious to reuse its elements within itself and other applications



With a modular approach

- Easy to keep track of components
- Single location to debug if something fails
- Structured cleanly for someone new to learn and contribute
- Element usability within itself and other applications

Next steps

If you are yet to learn shiny, start with modular approach

If you learned it the monolith way, start with converting an existing app modular

Check out Mastering Shiny book by Hadley Wickham

Check out the Github repo for modular bigfoot app

Deepsha Menghani: https://deepshamenghani.quarto.pub/dmenghani/