Welcome to Shiny Apps

- What is Shiny?
- - A web application framework for R.
- - Enables building interactive web apps using R code.

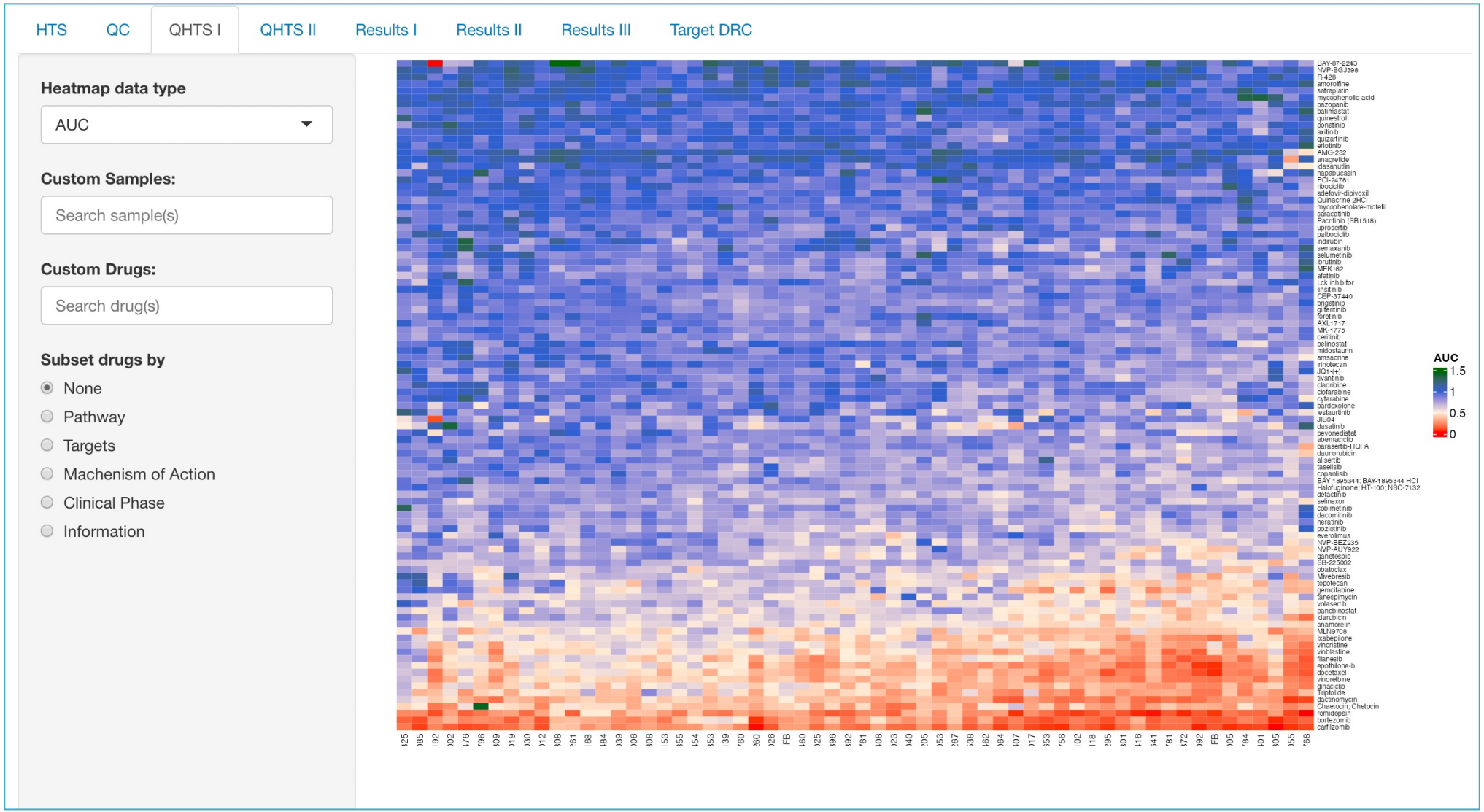
- Why Learn Shiny?
- - Create data dashboards, visualizations, and tools.
- - Share data insights interactively with non-programmers.

Components of a Shiny App

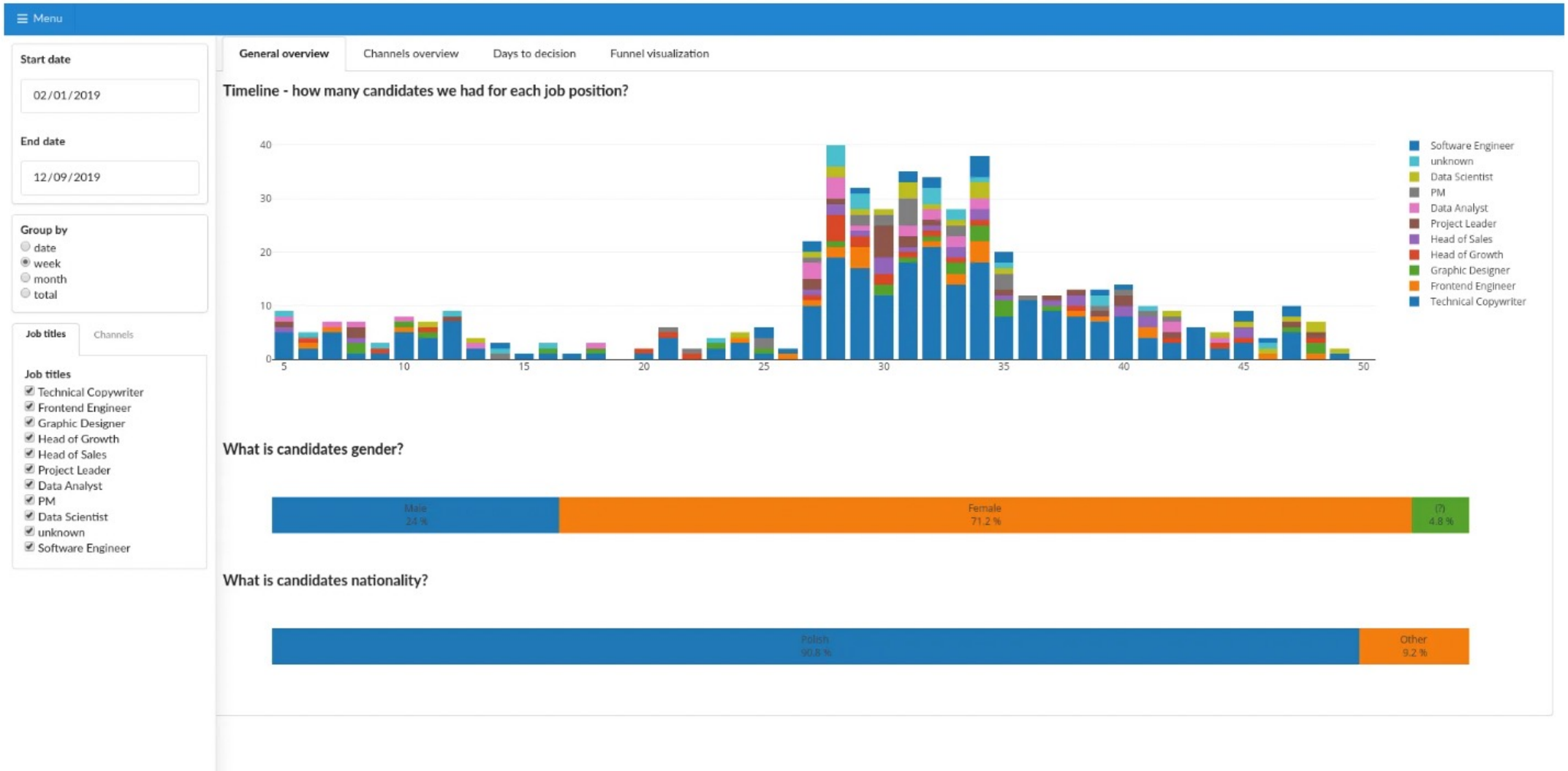
- **UI (User Interface):** Defines the layout and appearance.
- **Server:** Contains the logic to process inputs and generate outputs.
- **App:** Combines the UI and Server.

```r
library(shiny)

ui <- fluidPage(
  # UI elements
)

server <- function(input, output) {
  # Server logic
}

shinyApp(ui = ui, server = server)
```

Shiny App Layout - UI

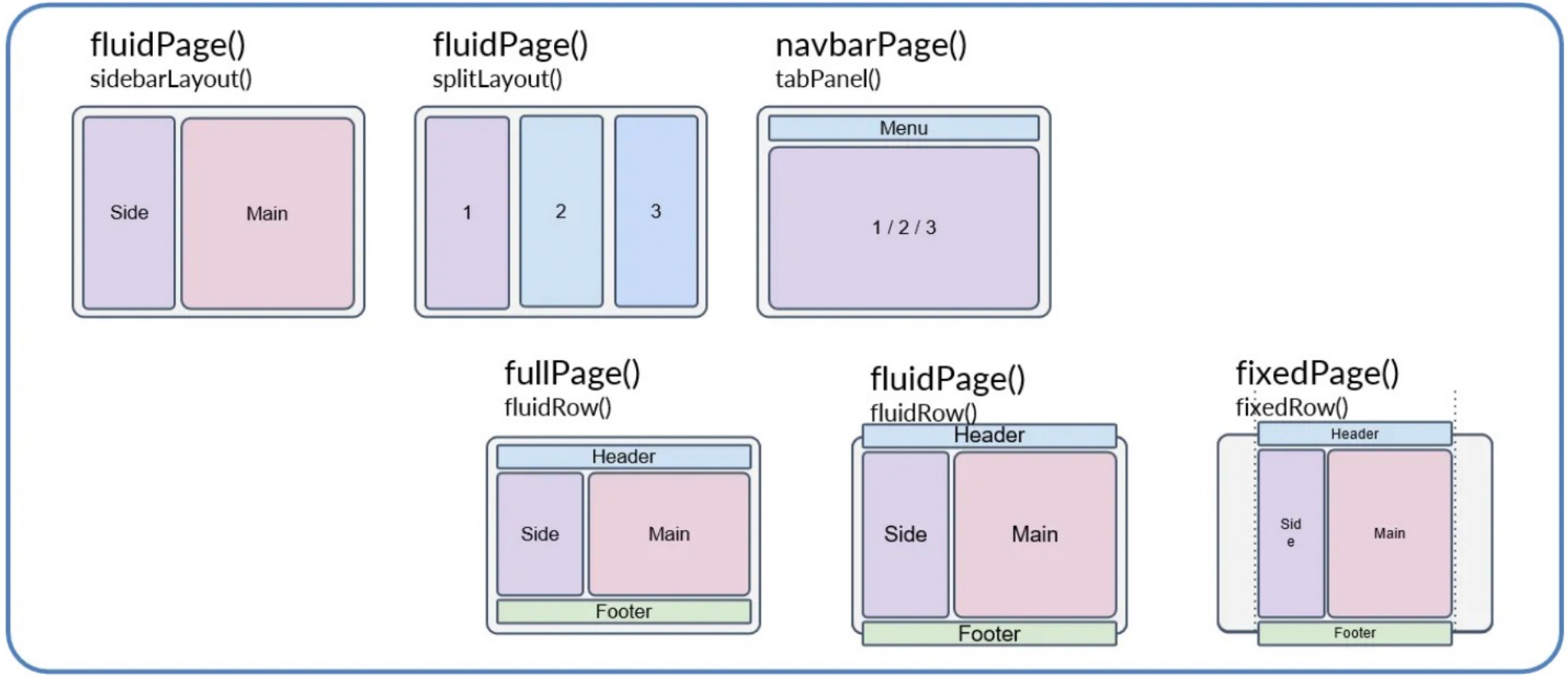


Shiny App Layout - UI



https://www.appsilon.com/post/journey-from-basic-prototype-to-production-ready-shiny-dashboard

Shiny App Layout - UI



https://www.appsilon.com/post/shiny-application-layouts

UI Components

- **Layout Functions:**

  - **Create a flexible layout**
    fluidPage()

  - **Define structured layouts**
  - sidebarLayout()
  - mainPanel()
  - sidebarPanel()

- **Input Functions:**

- sliderInput(), textInput(), selectInput(), etc.

- **Output Functions:**

- textOutput(), plotOutput(), tableOutput(), etc.

```
ui <- fluidPage(

titlePanel("Simple App"),

 sidebarLayout(

   sidebarPanel( selectInput("var", "Choose variable:",
                                          choices = c("A", "B", "C"))
   ),

   mainPanel( plotOutput("plot")
   )
 )
)
```

Server Logic

- How Server Works:
  - Reacts to user inputs.
  - Generates outputs dynamically.
- Key Functions:
  - renderText(), renderPlot(), renderTable()

Example Server Code:

```
server <- function(input, output) {
  output$plot <- renderPlot({
    plot(1:10, main = input$var)
  })
}
```

UI – Server connection

UI

```
ui <- fluidPage(

titlePanel("Simple App"),

  sidebarLayout(

    sidebarPanel( selectInput("var", "Choose
variable:",
                              choices = c("A", "B", "C"))
    ),

    mainPanel( plotOutput("plot")
)
  )
)
```

server

```
server <- function(input, output) {

  output$plot <- renderPlot({

    plot(1:10, main = input$var)

  })
}
```

Circular format

UI

```
selectInput(
  inputId = "colorVariable",
  label = "Please select variable to color point plot Fig1",
  choices = colnames(bCancer),
  selected = colnames(bCancer)[3],
  multiple = F


plotOutput("pointPlot"))
```

Circular format

UI

```
    selectInput(
      inputId = "colorVariable",
      label = "Please select variable to color point plot Fig1",
      choices = colnames(bCancer),
      selected = colnames(bCancer)[3],
      multiple = F


    plotOutput("pointPlot"))
```

server

```
    output$pointPlot <- renderPlot({
      xVariable = "concav" # Create these as constants right now
      yVariable = "area"

      bCancer %>%
        ggplot(aes_string(x = xVariable, y = yVariable, color = input$colorVariable)) + # Change color to
the input balue
        geom_point() +
        geom_smooth(method = "lm") +
        ggtitle("Fig1: Point plot") # Add title to identify which input it matches to
    })
```

Circular format

UI

```
selectInput(
  inputId = "colorVariable",
  label = "Please select variable to color point plot Fig1",
  choices = colnames(bCancer),
  selected = colnames(bCancer)[3],
  multiple = F

plotOutput("pointPlot"))
```

server

```
output$pointPlot <- renderPlot({
  xVariable = "concav" # Create these as constants right now
  yVariable = "area"

  bCancer %>%
    ggplot(aes_string(x = xVariable, y = yVariable, color = input$colorVariable)) + # Change color to
the input balue
    geom_point() +
    geom_smooth(method = "lm") +
    ggtitle("Fig1: Point plot") # Add title to identify which input it matches to
})
```

Circular format

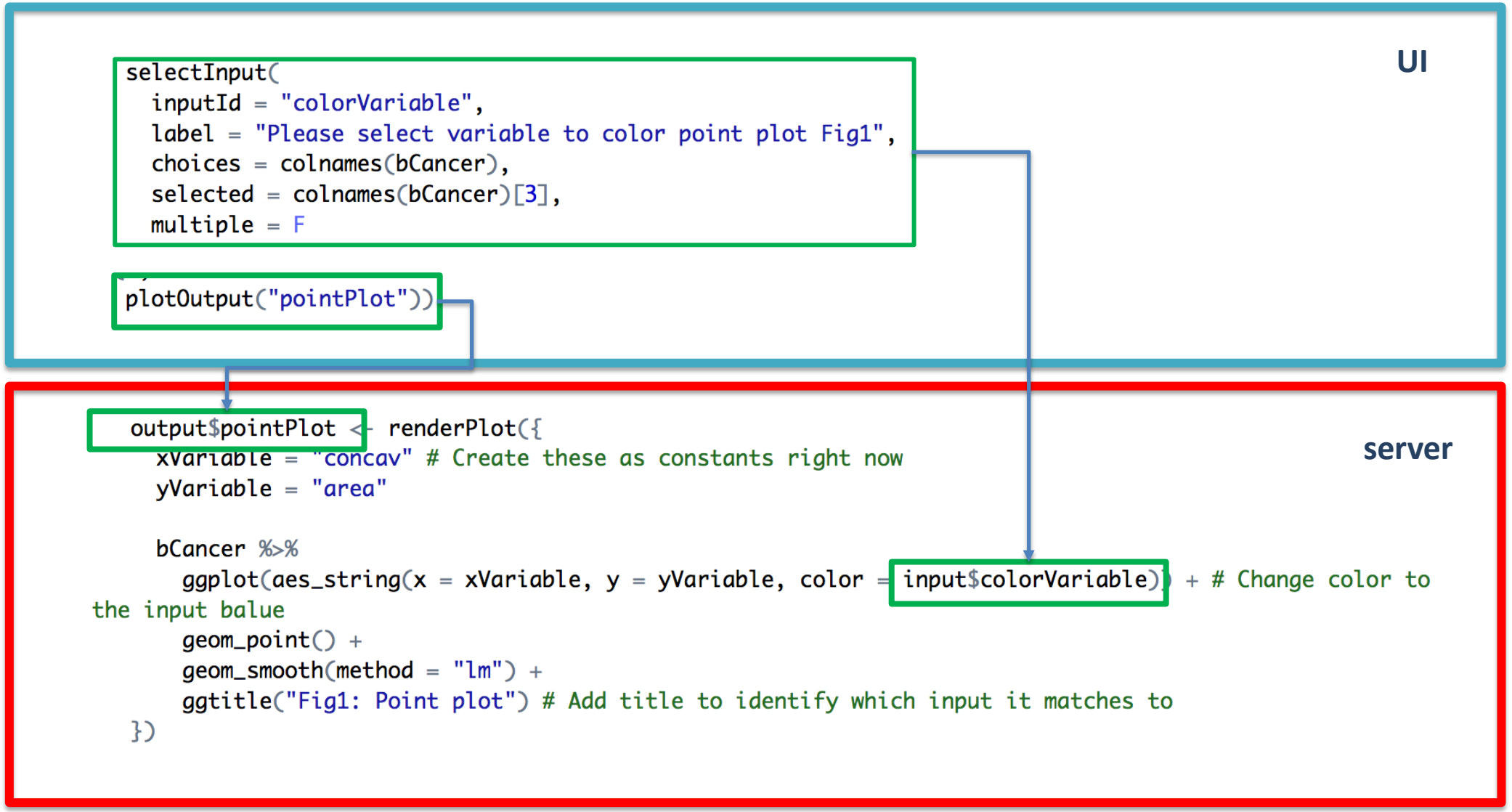
UI

```
selectInput(
  inputId = "colorVariable",
  label = "Please select variable to color point plot Fig1",
  choices = colnames(bCancer),
  selected = colnames(bCancer)[3],
  multiple = F

plotOutput("pointPlot"))
```

server

```
output$pointPlot <- renderPlot({
  xVariable = "concav" # Create these as constants right now
  yVariable = "area"

  bCancer %>%
    ggplot(aes_string(x = xVariable, y = yVariable, color = input$colorVariable)) + # Change color to
the input balue
    geom_point() +
    geom_smooth(method = "lm") +
    ggtitle("Fig1: Point plot") # Add title to identify which input it matches to
})
```

Circular format



Adding Interactivity

Reactive Expressions:

- Use reactive() for reusable calculations.

```
reactiveVal <- reactive({
  input$num * 2
})
```

Observe for Side Effects:

- Use observe() to perform actions without generating outputs.

```
observe({
  print(input$num)
})
```

When to Use Reactivity

- You should use reactivity whenever:

    1. **Dynamic Output Is Needed:** The content of your app (e.g., plots, tables, text) needs to change based on user inputs.
    2. **Dependent Computations Exist:** You want to compute something once and reuse the result across multiple outputs.
    3. **Efficient Updates Are Required:** You want to avoid recalculating unchanged components unnecessarily.

- Examples:
    - Displaying a plot that updates when a user changes a dropdown value.
    - Filtering a table based on user-selected criteria.
    - Dynamically displaying text based on slider input.

LET’S MAKE A SHINY APP!