

Interactive Web frameworks in R. Shiny, Plotly

R Nano Course Series

01/31/2016

Min Soo Kim
Bioinformatics Core Facility
Department of Bioinformatics
University of Texas Southwestern

Introduction

Interactive Analysis:

- High-dimensional data can be difficult to analyze.
- Interactive analysis and visualization can provide better understanding of variables and their significance.
- Explore relationship between multiple variables.

Web-based Content:

- View results beyond your own desktop.
- Share results with collaborators.
- Prepare web content for publication.

Introduction - Examples

Plotly Examples:

/Plotly_Examples/Example1/index.html

<https://plot.ly/r/dashboard/>

Shiny Examples:

<http://webpopix.org:8080/old/tgi1/>

Plotly - <https://plot.ly/>

Plotting libraries for R, Matlab, Python, and JavaScript.

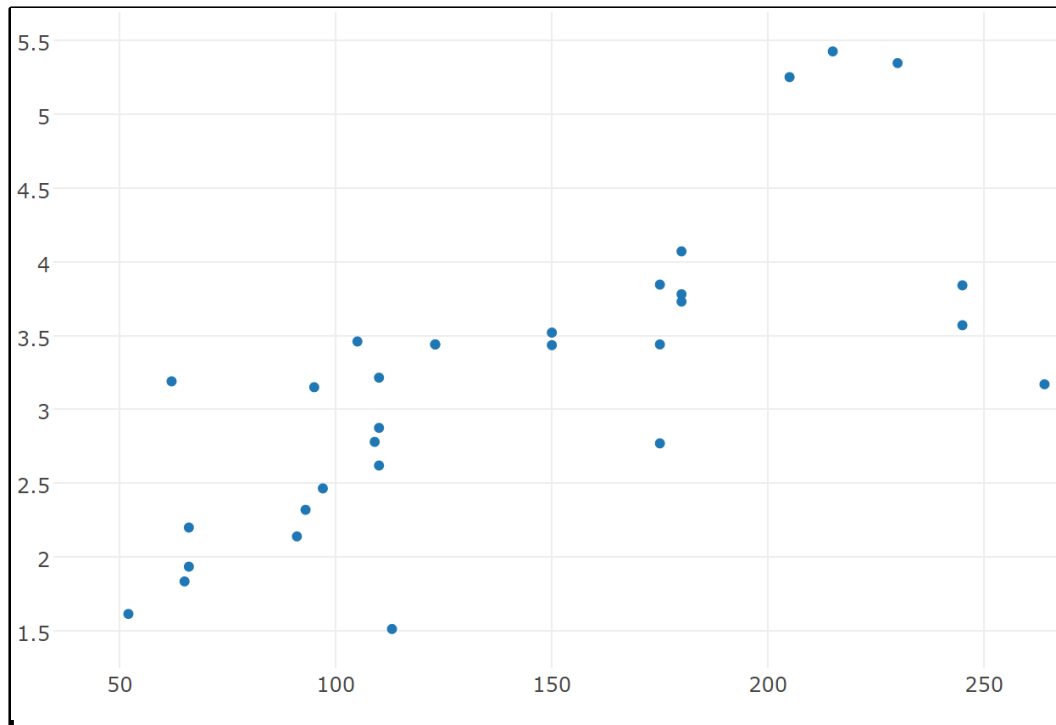
- Easy to use code for creating web-ready figures.
- Can be done with as little as one line of code.
- Available for free.
- Provides option to have your figures hosted on their cloud service for a fee. Payed version also gives access to online editing tool.

Plotly – Getting Started

```
install.packages("plotly")
```

```
library(plotly)
```

```
plot_ly(data=mtcars, x = mtcars$hp, y = mtcars$wt)
```



Plotly – index.html

- Figures generated by Plotly are output as an ‘index.html’ file which can be opened with any web browser.
- All data for the figure is stored in a self-contained folder that should be kept with the ‘index.html’ at all times.
- To make your figure available on the internet:
 1. Copy folder to your web hosting server.
 2. Create an account with Plotly for web publishing (fee).

Scatterplot

Why use Plotly instead of regular static figure?

1) Interactive Labels

2) Focus

```
plot_ly(data = diamonds, x = ~carat, y = ~price,  
        color = ~carat, size = ~carat,  
        text = ~paste("Price: $", price, '<br>Cut:', cut))
```

[\Plotly_Examples\ScatterPlot\index.html](#)

Box Plots

```
plot_ly(y = ~rnorm(50), type = "box") %>%  
  add_trace(y = ~rnorm(50, 1))
```

Note: %>% is used for combining plots.

[\Plotly_Examples\BoxPlot\index.html](#)

Density Plot

```
data <- density(mtcars$mpg)
plot_ly(x=data$x,y=data$y,mode = 'lines', fill = 'tozero')
```

[\Plotly_Examples\DensityPlot\index.html](#)

[\Plotly_Examples\RangePlot\index.html](#)

3D Plot

```
plot_ly(z = ~volcano, type = "surface")
```

[\Plotly_Examples\3dScatterPlot\index.html](#)

[\Plotly_Examples\SurfacePlot\index.html](#)

[\Plotly_Examples\3dCube\index.html](#)

Combine Plots

```
data <- density(mtcars$mpg)
```

```
p1 <- plot_ly(x = data$x, y = data$y, mode = 'lines')
```

```
p2 <- plot_ly(x = data$x, y = data$y, mode = 'lines', color = 'red')
```

```
subplot(p1, p2)
```

```
\Plotly_Examples\CombinedPlot\index.html
```

ggplotly

```
p <- ggplot(diamonds, aes(x = carat, y = price)) +  
  geom_point(aes(text = paste("Clarity:", clarity))) +  
  geom_smooth(aes(colour = cut, fill = cut)) +  
  facet_wrap(~ cut)
```

```
ggplotly(p)
```

```
\Plotly_Examples\ggplot\index.html
```

Reference

<https://plot.ly/r/reference/>

https://cpsievert.github.io/plotly_book/

<https://plot.ly/api/>

Shiny in R

- ❑ R package for creating Interactive Web Applications.
- ❑ Created by same developers who made Rstudio.
- ❑ No HTML, CSS, or JavaScript knowledge required.

Getting Started

```
install.packages("shiny")  
library(shiny)
```

Iris

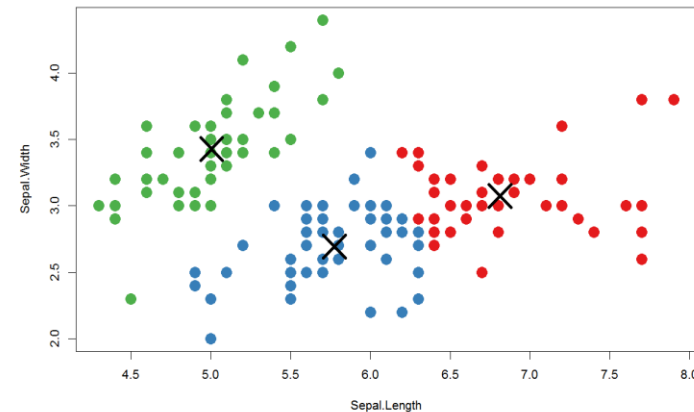
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3	1.4	0.1	setosa
4.3	3	1.1	0.1	setosa

Iris k-means clustering

X Variable
Sepal.Length

Y Variable
Sepal.Width

Cluster count
3



\\Shiny_Examples\\K-means\\app.R

Basic Template

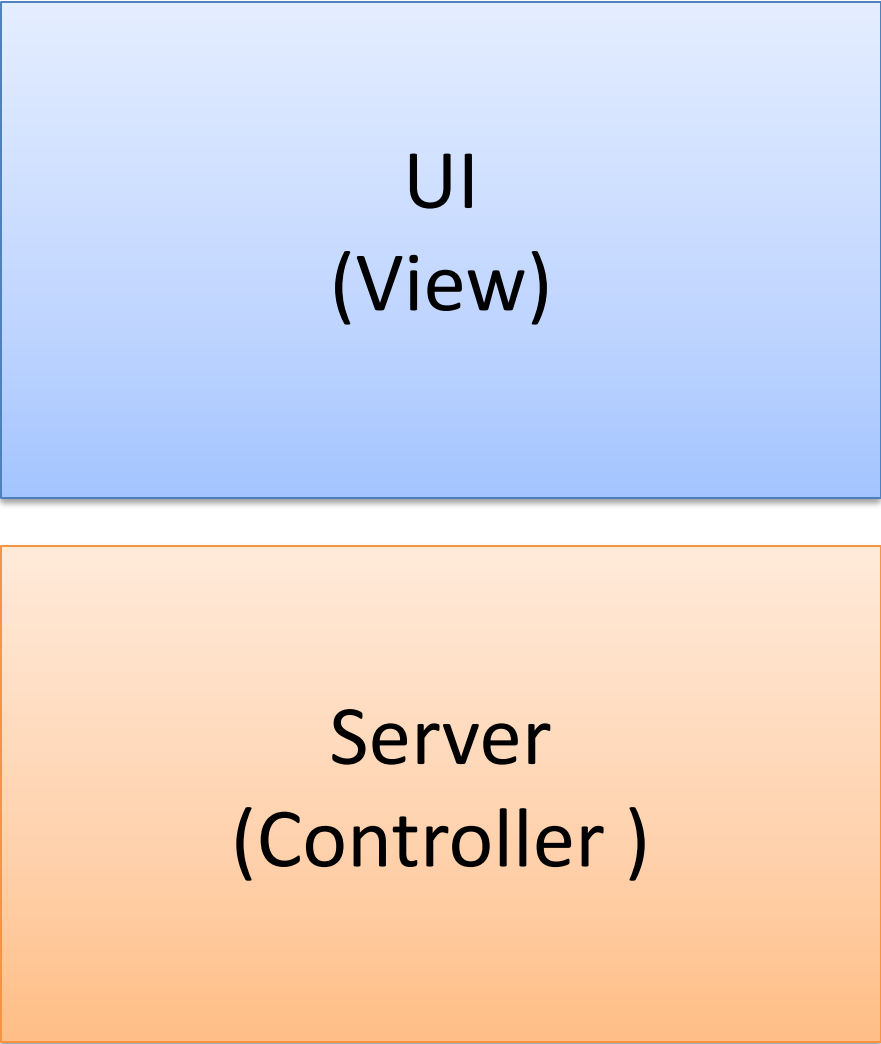
```
library(shiny)
```

```
ui <- fuilPage( )
```

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

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


View (Front) / Controller (Back)



The diagram illustrates the View/Controller architecture. It consists of two main components: a 'UI (View)' represented by a light blue rectangle at the top, and a 'Server (Controller)' represented by a light orange rectangle at the bottom. To the right of the UI box is the code snippet 'ui <- fuilPage()'. To the right of the Server box is the code snippet 'server <- function(input,output){ }'. There are no explicit arrows or lines connecting the two boxes, suggesting a clear separation of concerns between the front-end and back-end.

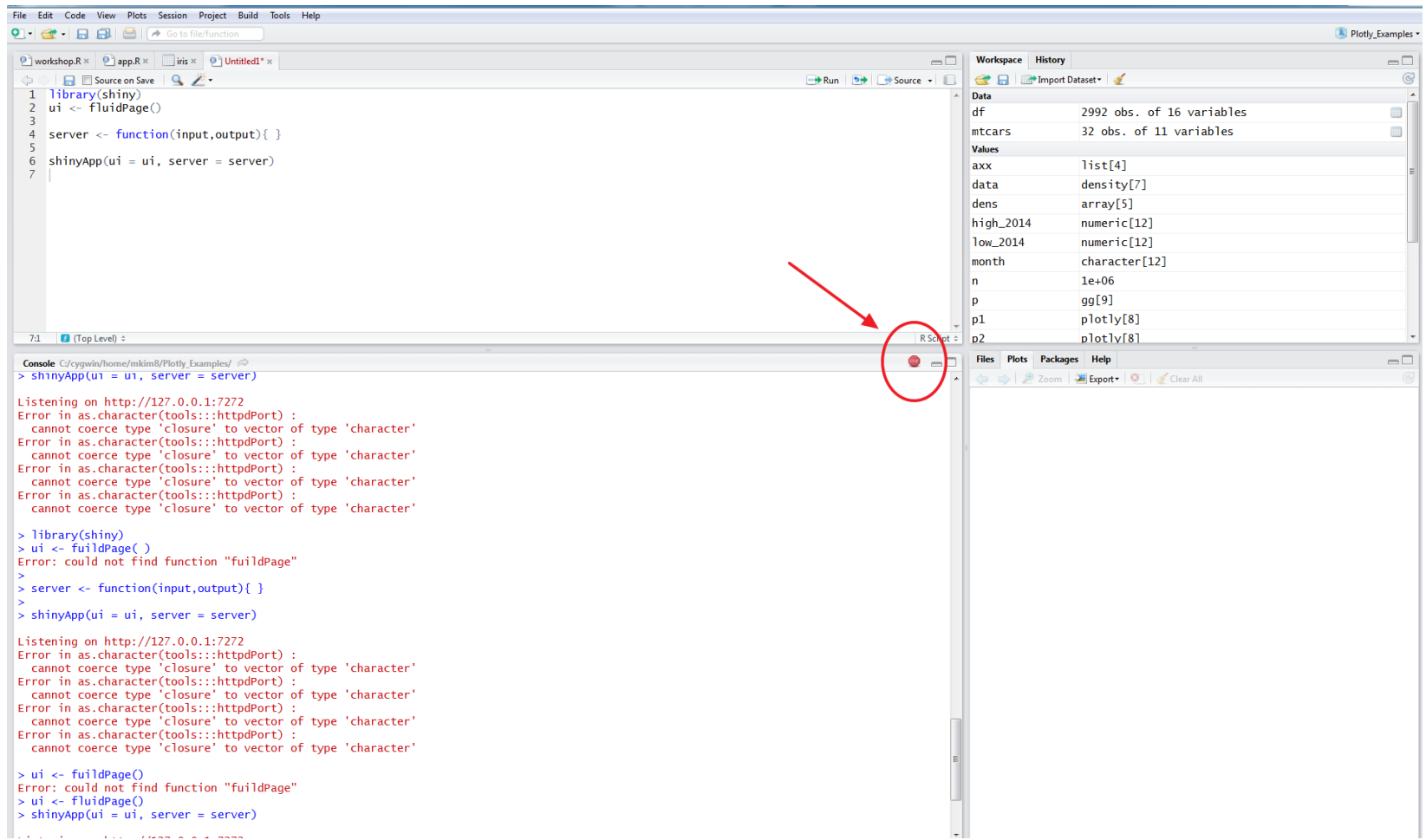
UI
(View)

```
ui <- fuilPage( )
```

Server
(Controller)

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

Close the app



The screenshot shows the RStudio IDE with a Shiny application. The console displays several error messages related to coercion and function finding. A red circle highlights the 'R Script' button in the top right corner, with a red arrow pointing to it.

Console Output:

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

Listening on http://127.0.0.1:7272
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'

> library(shiny)
> ui <- fuilPage()
Error: could not find function "fuilPage"
>
> server <- function(input,output){ }
>
> shinyApp(ui = ui, server = server)

Listening on http://127.0.0.1:7272
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'
Error in as.character(tools:::httpdPort) :
  cannot coerce type 'closure' to vector of type 'character'

> ui <- fuilPage()
Error: could not find function "fuilPage"
> ui <- fluidPage()
> shinyApp(ui = ui, server = server)
```

Workspace:

Object	Class	Attributes
df	data.frame	2992 obs. of 16 variables
mtcars	data.frame	32 obs. of 11 variables
axx	list	list[4]
data	density	density[7]
dens	array	array[5]
high_2014	numeric	numeric[12]
low_2014	numeric	numeric[12]
month	character	character[12]
n	integer	1e+06
p	gg	gg[9]
p1	plotly	plotly[8]
p2	plotly	plotly[8]

First element to UI

```
library(shiny)
```

```
ui <- fluidPage("Hello World")
```

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

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

UI Elements

Iris K-means clustering

X Variable

Sepal.Length

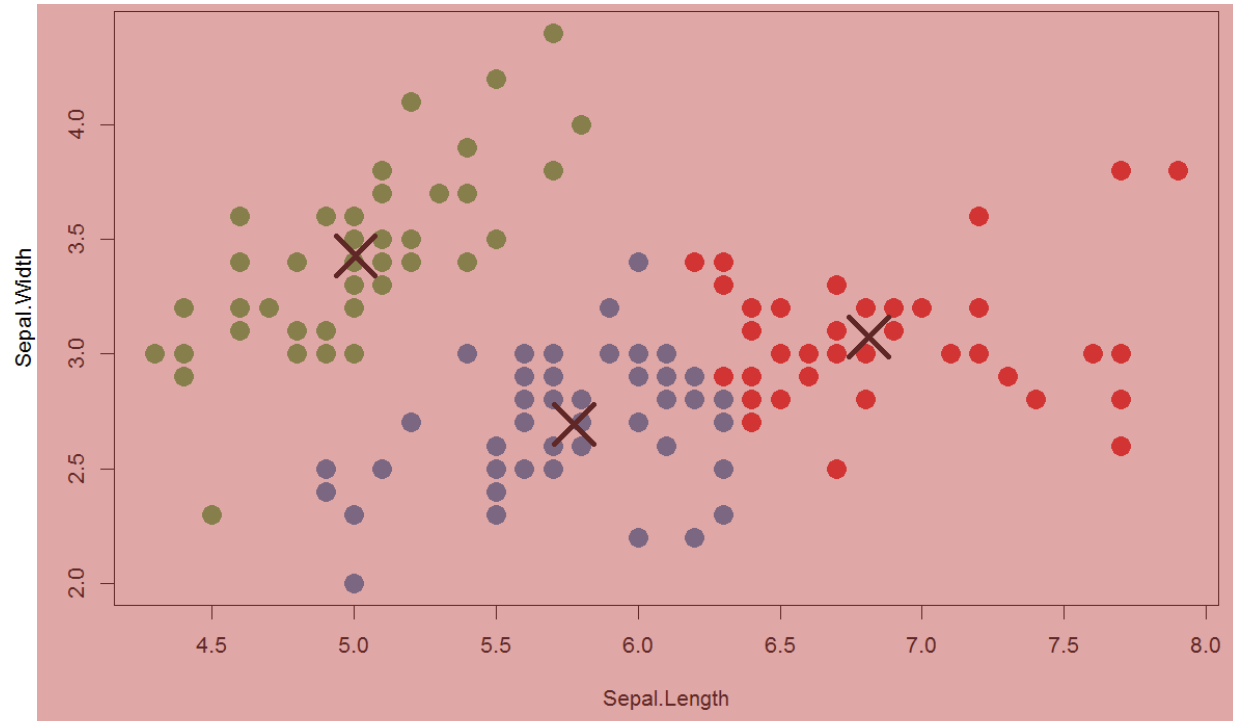
Y Variable

Sepal.Width

Cluster count

3

Input Elements



Output Elements

UI Elements :

Inputs and Outputs

```
ui <- fluidPage(  
  headerPanel("Iris k-means clustering"),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris), selected = names(iris)[[2]]),  
    numericInput('clusters', 'Cluster count', 3, min = 1, max = 9)  
  ),  
  mainPanel(  
    plotOutput('plot1')  
  )  
)
```

Iris K-means clustering

X Variable

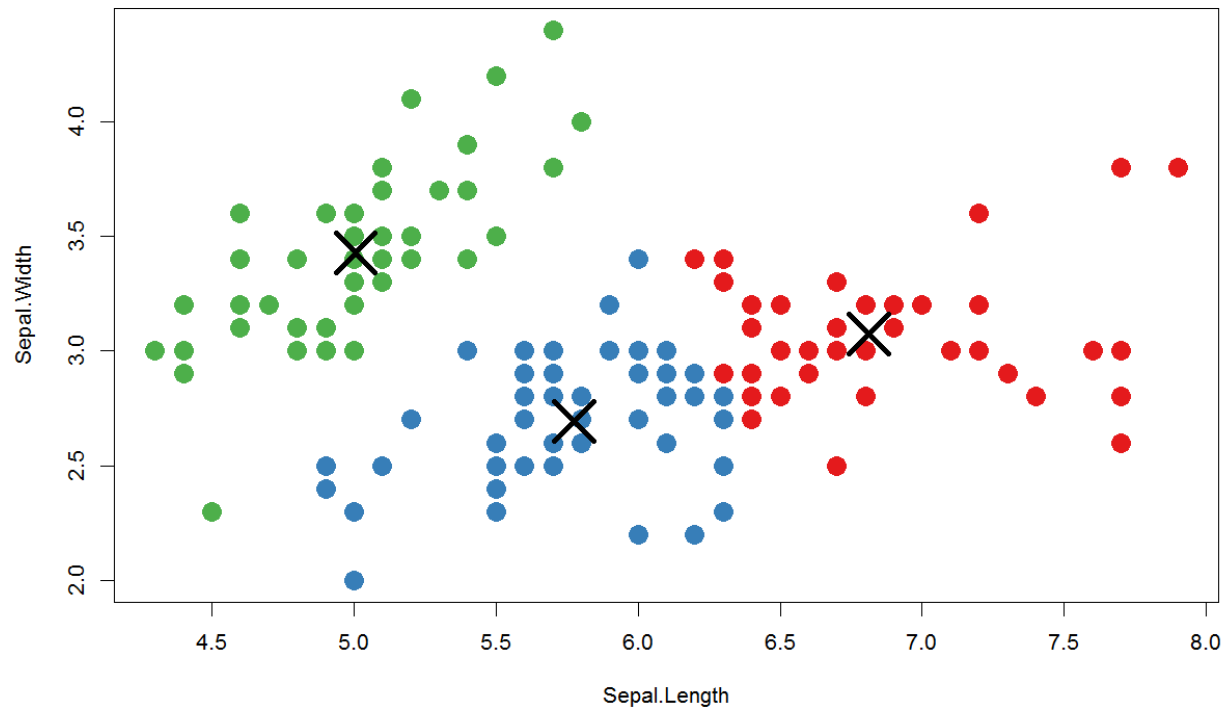
Sepal.Length

Y Variable

Sepal.Width

Cluster count

3



UI Elements :

Inputs and Outputs

```
ui <- fluidPage(  
  headerPanel("Iris k-means clustering"),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris), selected = names(iris)[[2]]),  
    numericInput('clusters', 'Cluster count', 3, min = 1, max = 9)  
  ),  
  mainPanel(  
    plotOutput('plot1')  
  )  
)
```

Iris K-means clustering

X Variable

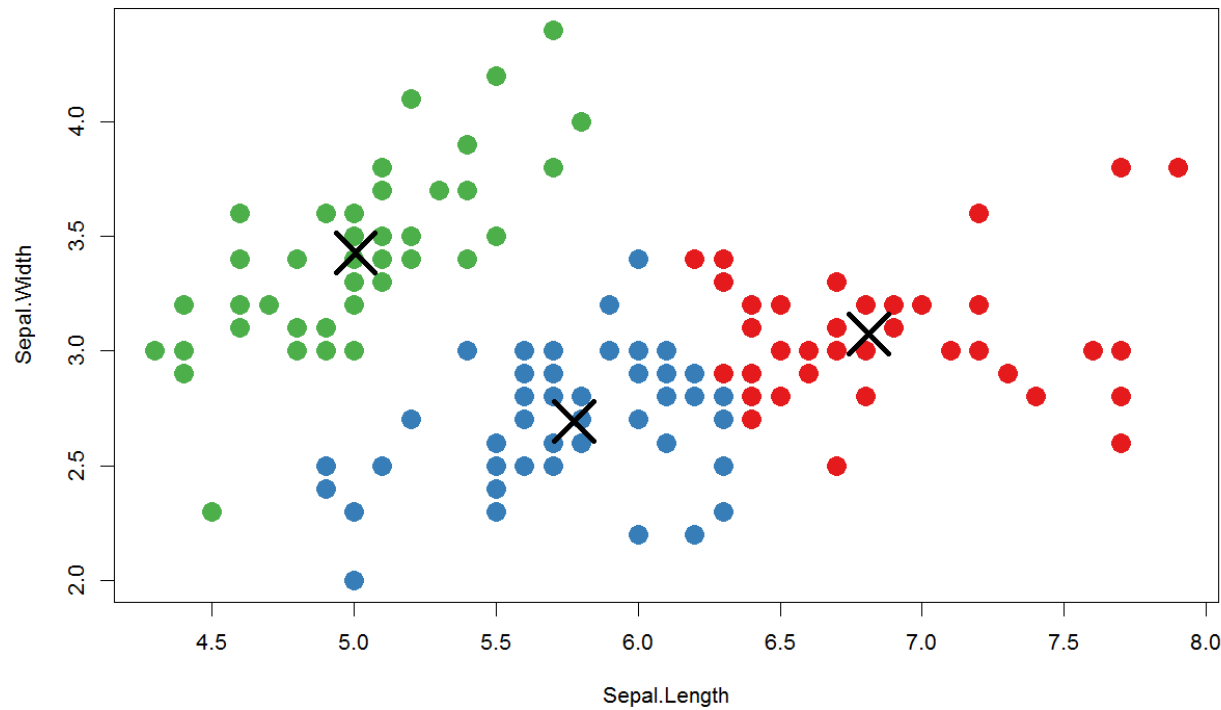
Sepal.Length

Y Variable

Sepal.Width

Cluster count

3



UI Element : Input

```
selectInput( 'xcol' , "X Variable" , names(iris) )
```

input name
(reference)

Display Label

Input Arguments

Iris K-means clustering

X Variable

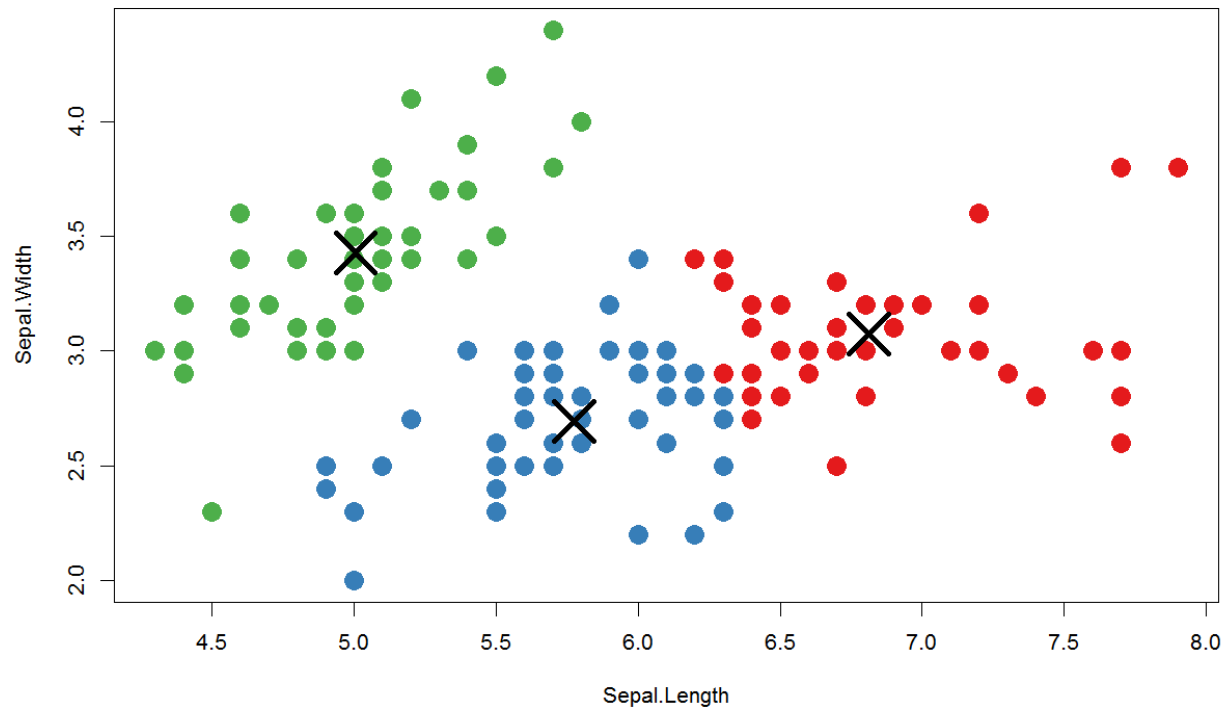
Sepal.Length

Y Variable

Sepal.Width

Cluster count

3



Buttons

Action

Submit

`actionButton()`
`submitButton()`

Single checkbox

☒ Choice A

`checkboxInput()`

Checkbox group

☒ Choice 1
☐ Choice 2
☐ Choice 3

`checkboxGroupInput()`

Date input

2014-01-01

`dateInput()`

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

Choose File No file chosen

`fileInput()`

Numeric input

1

`numericInput()`

Password Input

.....

`passwordInput()`

Radio buttons

☒ Choice 1
☐ Choice 2
☐ Choice 3

`radioButtons()`

Select box

Choice 1

`selectInput()`

Sliders

0 50 100
0 25 75 100

`sliderInput()`

Text input

Enter text...

`textInput()`

© CC 2015 RStudio, Inc.

<http://shiny.rstudio.com/reference/shiny/latest/>

UI Elements :

Inputs and Outputs

```
ui <- fluidPage(  
  headerPanel("Iris k-means clustering"),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris), selected = names(iris)[[2]]),  
    numericInput('clusters', 'Cluster count', 3, min = 1, max = 9)  
  ),  
  mainPanel(  
    plotOutput('plot1')  
  )  
)
```

UI Element : Output

Iris K-means clustering

X Variable

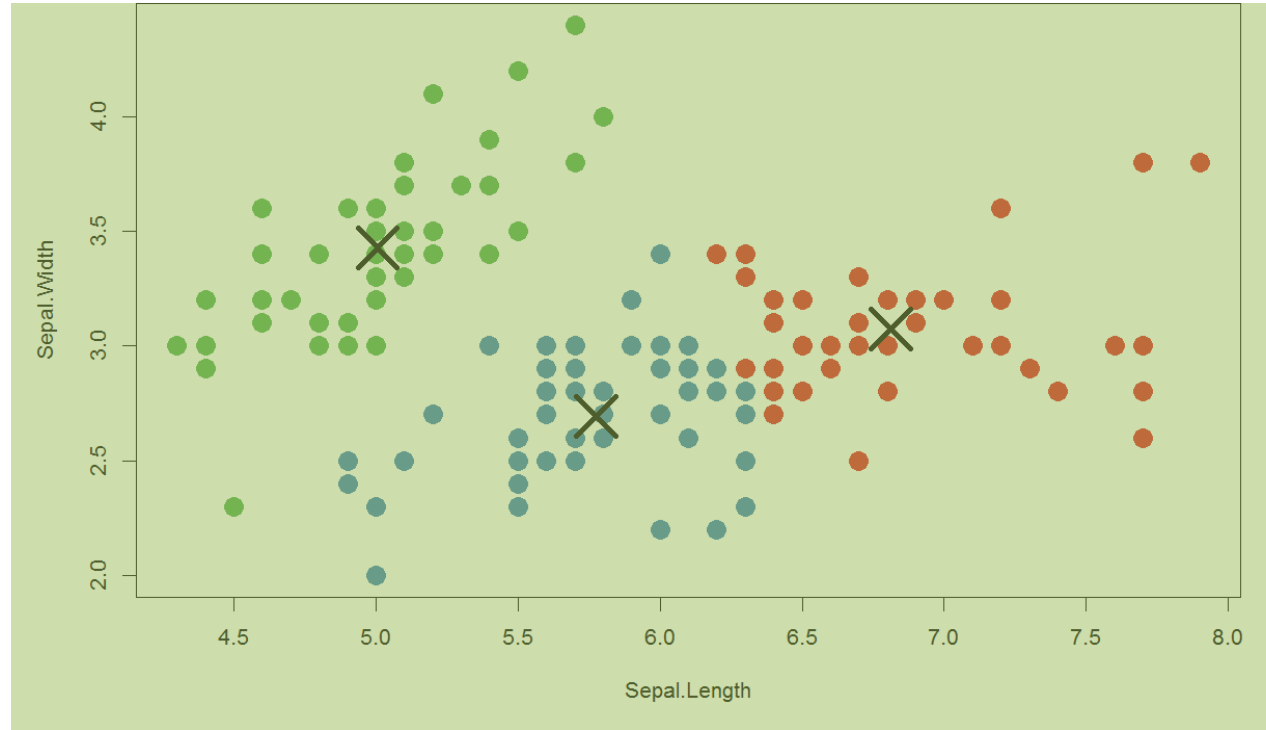
Sepal.Length

Y Variable

Sepal.Width

Cluster count

3



```
ui <- fluidPage(  
  headerPanel("Iris k-means clustering"),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris), selected = names(iris)[[2]]),  
    numericInput('clusters', 'Cluster count', 3, min = 1, max = 9)  
  ),  
  mainPanel(  
    plotOutput('plot1')  
  )  
)
```

UI Element : Output

plotOutput('plot1')

Output Type

input name
(object reference)

UI Element : Output

Function	Inserts
<code>dataTableOutput()</code>	an interactive table
<code>htmlOutput()</code>	raw HTML
<code>imageOutput()</code>	image
<code>plotOutput()</code>	plot
<code>tableOutput()</code>	table
<code>textOutput()</code>	text
<code>uiOutput()</code>	a Shiny UI element
<code>verbatimTextOutput()</code>	text

<http://shiny.rstudio.com/reference/shiny/latest/>

UI Elements :

Layout

```
ui <- fluidPage(  
  headerPanel("Iris k-means clustering"),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris), selected = names(iris)[[2]]),  
    numericInput('clusters', 'Cluster count', 3, min = 1, max = 9)  
  ),  
  mainPanel(  
    plotOutput('plot1')  
  )  
)
```

Without Layout

Iris k-means clustering

X Variable

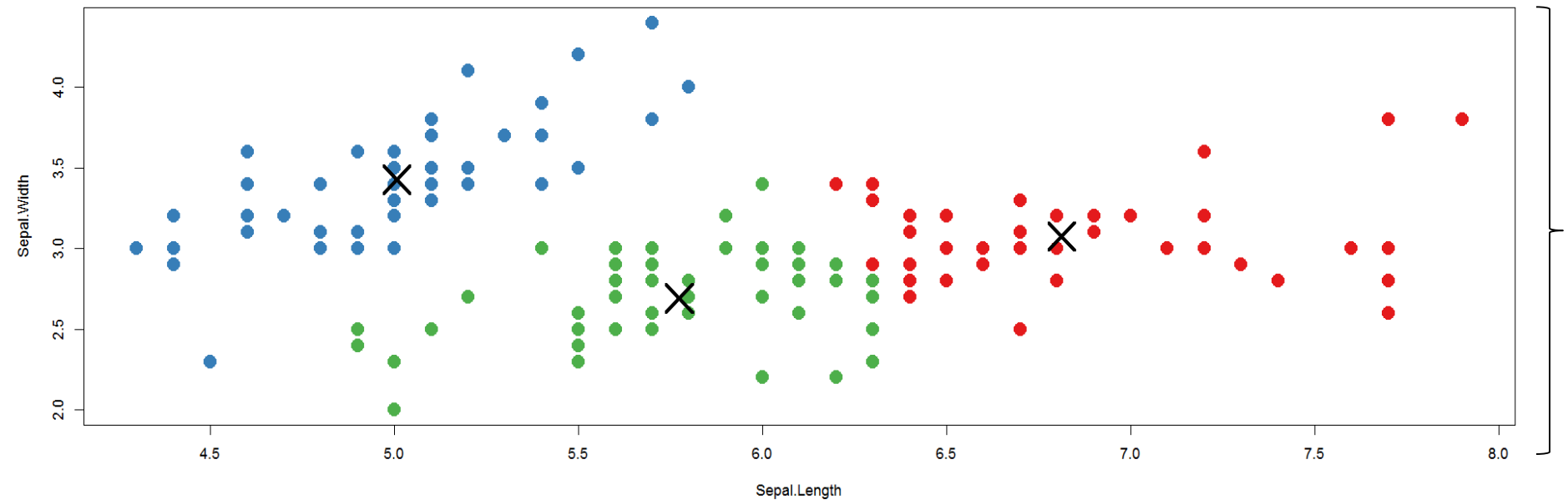
Sepal.Length

Y Variable

Sepal.Width

Cluster count

3



Layout

Iris K-means clustering

headerPanel()

X Variable

Sepal.Length

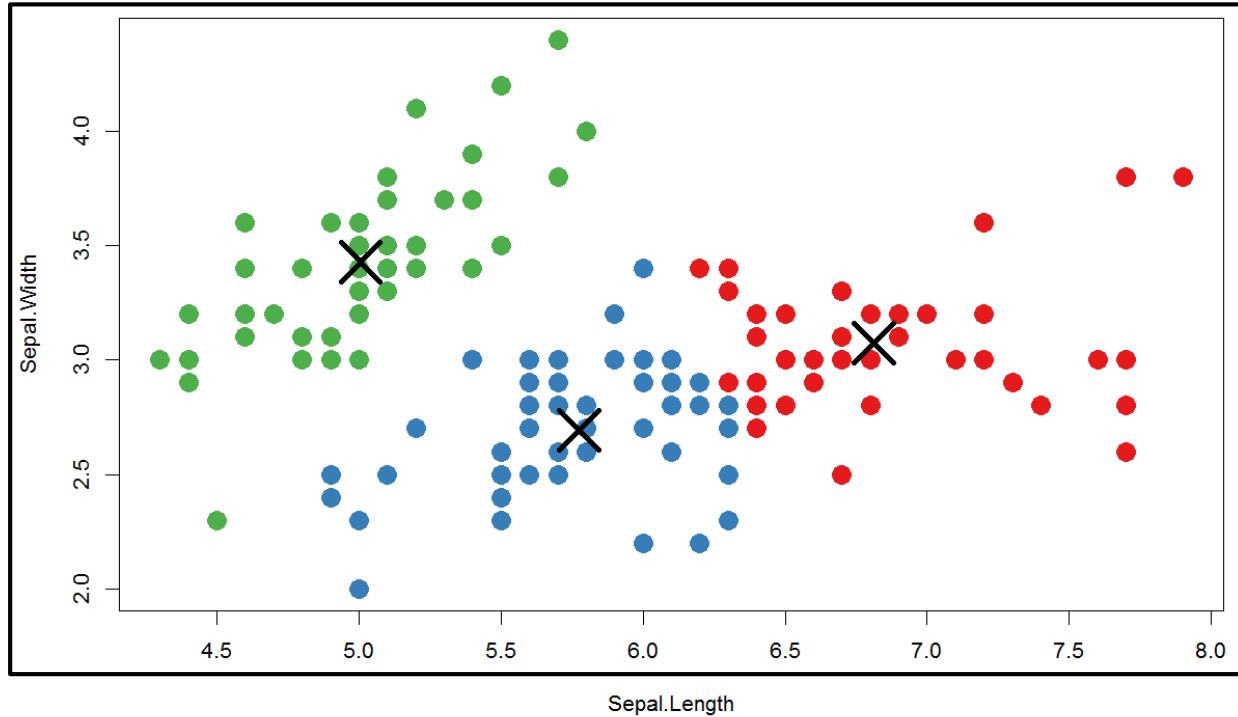
Y Variable

Sepal.Width

Cluster count

3

sidebarPanel()



mainPanel()

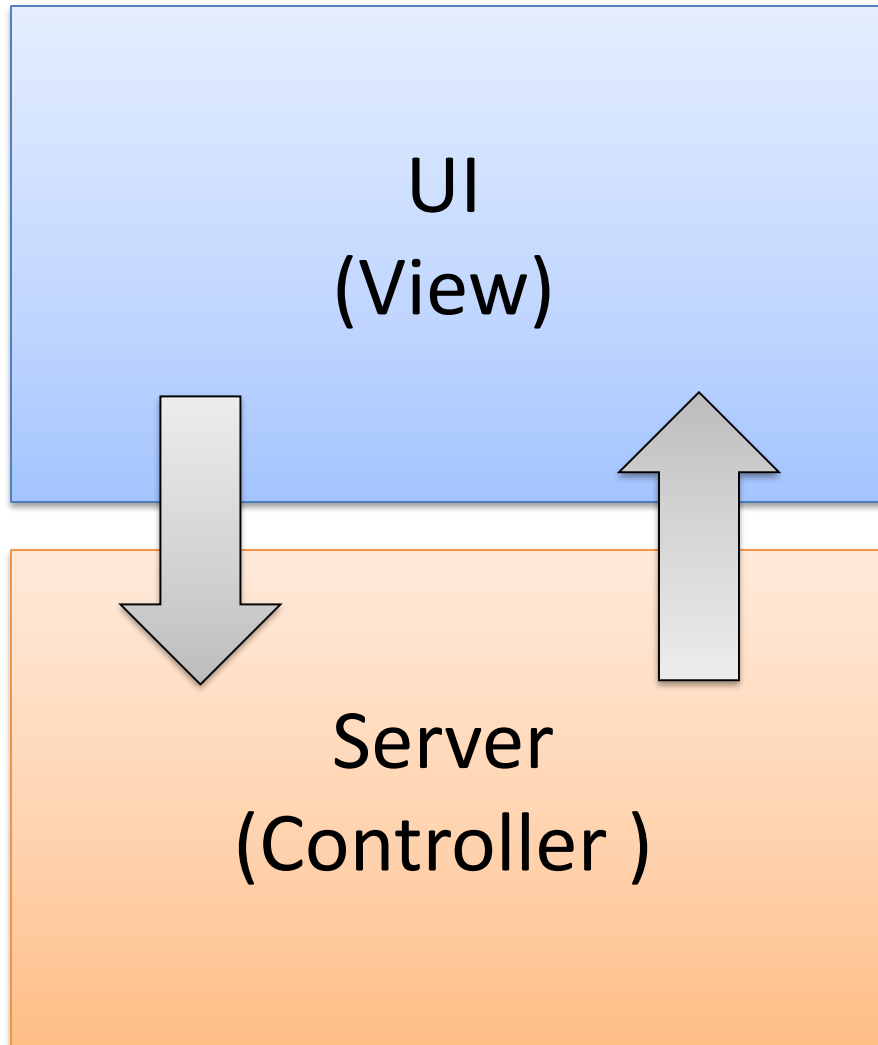
UI Elements: Commas

```
ui <- fluidPage(  
  headerPanel("Iris k-means clustering") ,  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)) ,  
    selectInput('ycol', 'Y Variable', names(iris),selected = names(iris)[[2]]) ,  
    numericInput('clusters', 'Cluster count', 3, min = 1, max = 9)  
  ) ,  
  mainPanel(  
    plotOutput('plot1')  
  )  
)
```

UI Elements

```
ui <- fluidPage(  
  headerPanel("Iris k-means clustering"),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris), selected = names(iris)[[2]]),  
    numericInput('clusters', 'Cluster count', 3, min = 1, max = 9)  
  ),  
  mainPanel(  
    plotOutput('plot1')  
  )  
)
```

View (Front) / Controller (Back)



```
ui <- fuilddPage( )
```

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

Template

```
library(shiny)
```

```
ui <- fuilPage( )
```

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

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

Server Elements: Output

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

```
  output$plot1 <- renderPlot({
```

```
    par(mar = c(5.1, 4.1, 0, 1))
```

```
    plot(selectedData(), col = clusters()$cluster, pch = 20, cex = 3)
```

```
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
```

```
  })
```

```
}
```


What gets sent to UI

output\$plot1



plotOutput('plot1')

Use `render*()` to build objects

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

```
  output$plot1 <- renderPlot({
```

```
    par(mar = c(5.1, 4.1, 0, 1))
```

```
    plot(selectedData(), col = clusters()$cluster, pch = 20, cex = 3)
```

```
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
```

```
  })
```

```
}
```

Use the **render*()** function that creates the type of output you wish to make.

function	creates
<code>renderDataTable()</code>	An interactive table <small>(from a data frame, matrix, or other table-like structure)</small>
<code>renderImage()</code>	An image (saved as a link to a source file)
<code>renderPlot()</code>	A plot
<code>renderPrint()</code>	A code block of printed output
<code>renderTable()</code>	A table <small>(from a data frame, matrix, or other table-like structure)</small>
<code>renderText()</code>	A character string
<code>renderUI()</code>	a Shiny UI element

Server Elements: Output

```
server <- function( input , output ) {  
  output$plot1 <- renderPlot({  
    plot( )  
  })  
}
```

The diagram highlights two parts of the R code with boxes and labels:

- A box around `renderPlot({` is labeled "Type of Object".
- A box around `plot()` is labeled "Code to build".

Server Elements: Output

```
server <- function( input , output ) {  
  output$plot1 <- renderPlot({
```

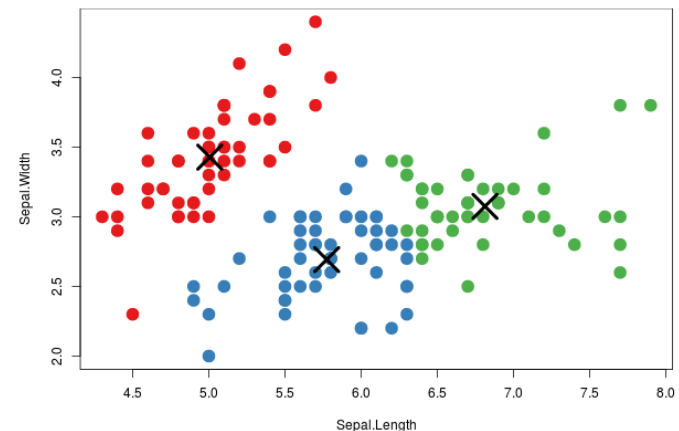
```
    par(mar = c(5.1, 4.1, 0, 1)) #Determines size of plot.
```

```
    plot(selectedData(), col = clusters()$cluster, pch = 20, cex = 3)
```

```
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
```

```
  })
```

```
}
```



UI Elements

```
ui <- fluidPage(  
  headerPanel("Iris k-means clustering"),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris), selected = names(iris)[[2]]),  
    numericInput('clusters', 'Cluster count', 3, min = 1, max = 9)  
  ),  
  mainPanel(  
    plotOutput('plot1')  
  )  
)
```

Server Elements: Input

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

```
  selectedData <- reactive({
```

```
    iris[ , c( input$xcoll , input$ycol )]
```

```
  })
```

```
}
```



Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3	1.4	0.1	setosa
4.3	3	1.1	0.1	setosa

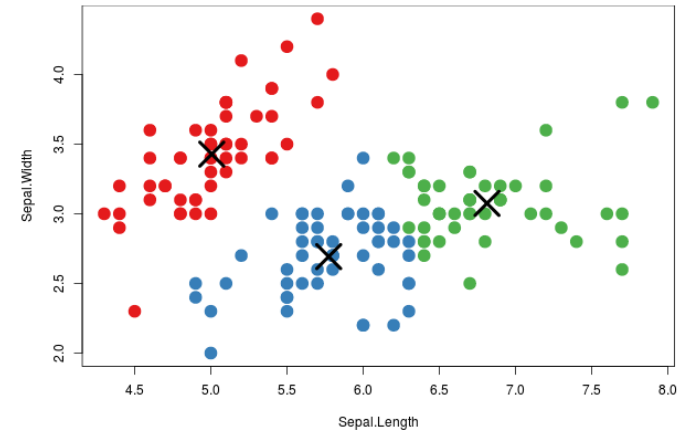
Use `reactive({ })`

How does the 'server' know changes were made to the input variables?

Using `reactive({ input$variable })` tells server this a variable that will be dynamically updated.

Server Elements: Input

```
server <- function( input , output ) {  
  
  selectedData <- reactive({  
    iris[ , c( input$xcol , input$ycol )]  
  })  
  
  clusters <- reactive({  
    kmeans(selectedData(), input$clusters)  
  })  
  
}
```



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

```
  selectedData <- reactive({  
    iris[, c(input$xcol, input$ycol)]  
  })
```

```
  clusters <- reactive({  
    kmeans(selectedData(), input$clusters)  
  })
```

```
  output$plot1 <- renderPlot({  
    par(mar = c(5.1, 4.1, 0, 1))  
    plot(selectedData(), col = clusters()$cluster, pch = 20, cex = 3)  
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)  
  })
```

```
}
```

Basic Template

```
ui <- fuilPage( )
```

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

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

ui.R and server.R

```
ui <- fluidPage( )
```

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

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

ui.R

```
library(shiny)
fluidPage(
  sliderInput()
  plotOutput()
)
```

server.R

```
library(shiny)
function(input, output) {
  output$plot <- renderPlot({})
}
)
```

Sharing Online

Option 1 – Private Web Server

Shiny Server :

www.rstudio.com/products/shiny/shiny-server/

Option 2 – shinyapps.io

Publish Shiny Apps Online :

<http://shiny.rstudio.com/articles/shinyapps.html>

FREE	BASIC	STANDARD
\$0 /month	\$39 /month (or \$440/year)	\$99 /month (or \$1,100/year)
<small>New to Shiny? Deploy your applications to the cloud for FREE. Perfect for teachers and students or those who want a place to learn and play. No credit card required.</small>	<small>Take your users' experience to the next level. shinyapps.io Basic lets you scale your application performance by adding R processes dynamically as usage increases.</small>	<small>Need password protection? shinyapps.io Standard lets you authenticate your application users.</small>
5 Applications	Unlimited Applications	Unlimited Applications
25 Active Hours	250 Active Hours	1000 Active Hours
✓ Community Support	✓ Multiple Instances	✓ Authentication
ⓘ RStudio Branding	✓ Email Support	✓ Multiple Instances
		✓ Email Support

References

<http://shiny.rstudio.com/tutorial/>

<http://shiny.rstudio.com/reference/shiny/latest/>

<http://shiny.rstudio.com/articles/>