Shiny\_ResBazSydney2018 notes

**Lesson 1**

*# ResBaz Sydney 2018 Advanced R Session - Get interactive with Shiny*

*# Lesson 1 - under the hood of a Shiny app*

*install.packages('shiny')*

*library(shiny)*

*# these instructive examples come with the Shiny package*

*# check out https://shiny.rstudio.com/articles/basics.html for a guide to the code*

*runExample("01\_hello")*

*# 01\_hello shows a basic example of reactivity: one input, one output*

*# - User changes the position of the slider, and this alters the binwidth of the histogram*

*# - The user interface updates pretty quickly, gives a seamless experience*

*#*

*# UI – user interface*

*# - Input widgets collect information from the user*

*# - Displays outputs, such as plots or tables*

*# - Define how visual elements are laid out*

*# - Written in R syntax, but most functions won’t be familiar*

*#*

*# Server*

*# - Where data is processed*

*# - Regular R code combined with special shiny:: functions that implement reactivity*

*# - Reactive expressions are re-evaluated when their dependent values have changed*

*#*

*# The sliderInput widget defined in the UI allows the user to set a value, which is retrieved and used to set the binwidth in the Server.*

*#*

*# renderPlot is a reactive expression: code wrapped within renderPlot({}) will be re-evaluated any time an input changes*

*#*

*# Exercise 1: identify how the value obtained by the sliderInput is used by the renderPlot expression in the Server*

*#*

*# Exercise 2: identify how the histogram object produced in the Server is passed to the User Interface*

*#*

*# N.B. if you want to use 'reactive values' (i.e. those that are updated), you must use them within one of Shiny's reactive expressions*

*runExample("02\_text")*

*# 02\_text uses multiple input widgets and displays multiple outputs*

*# this example also implements a 'chain of reactivity', using the reactive({}) expression*

*# reactive({}) allows us to assign an expression to an object, which can be called upon in other reactive expressions*

*#*

*# Exercise 3: trace the path from the "dataset" input in the UI through the Server, and back to the "view" table in the UI*

*# (hint, think in terms of inputs and outputs)*

*#*

*# Exercise 4: play around with resizing the Shiny application window - notice what happens to the layout*

**Lesson 2**

# Exercise 1: fill out the template to create a working app!

# Exercise 2: add functionality to select a 'factor' variable to colour points or change point shapes

library(shiny)

library(ggplot2)

ui <- fluidPage(

# Application title

titlePanel(""), # give our app a title

sidebarLayout( # this is Shiny's basic type of layout - more complex layouts exist

# Sidebar with input widgets

sidebarPanel(

## add x-axis selector

selectInput(inputID="x\_axis", label="Select x-axis", choices=c("Miles per gallon" = "mpg",

"Weight" = "wt",

"Horsepower" = "hp",

"Engine displacement" = "disp")),

## add y-axis selector

## add factor selector

## display a table

tableOutput("")

),

# Show a graph of the selected relationships in a 'main panel'

mainPanel(

## display the graph

plotOutput("")

)

)

)

# Define server logic required to draw a histogram

server <- function(input, output) {

# code to generate the table

output$YOURVALUEHERE <- renderTable({

# create a dataframe from the mtcars dataset with just the columns selected by the user using the selectInput()'s

})

# code to generate the plot

output$YOURVALUEHERE <- renderPlot({

ggplot(mtcars, aes\_string(x = input$x\_axis, y = input$y\_axis)) + # aes\_string is important here!

geom\_point() +

scale\_color\_gradientn(colours = rainbow(5))+

theme\_minimal()

})

}

# Run the application

shinyApp(ui = ui, server = server)

Lesson 3

# Exercise 1: use the help for updateSelectedInput() and work out how to preselect some values (rather than have them start all the same)

# Exercise 2: use the help for fileUpload() and work out how to add the file name as a title to the plot - in ggtitle()

library(shiny)

library(ggplot2)

library(plotly)

ui <- fluidPage(

# Application title

titlePanel("Interactive data explorer"), # give our app a title

sidebarLayout( # this is Shiny's basic type of layout - more complex layouts exist

# Sidebar with input widgets

sidebarPanel(

## add fileInput widget

fileInput("uploadedfile", "Choose CSV File",

multiple = FALSE,

accept = c("text/csv",

"text/comma-separated-values,text/plain",

".csv")),

## add x-axis selector

selectInput("x\_axis", "Select x-axis", choices=NULL),

## add y-axis selector

selectInput("y\_axis", "Select y-axis", choices=NULL),

## add factor selector

selectInput("fac", "Select factor", choices=NULL),

## display a table

tableOutput('table')

),

# Show a graph of the selected relationships in a 'main panel'

mainPanel(

## display the graph

plotlyOutput('plot') # plotlyOutput

)

)

)

# Define server logic required to draw a histogram

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

uploaded\_file <- eventReactive(input$uploadedfile, {

df <- read.csv(input$uploadedfile$datapath)

vars <- names(df)

updateSelectInput(session, "x\_axis", choices = vars)

updateSelectInput(session, "y\_axis", choices = vars)

updateSelectInput(session, "fac", choices = vars)

return(df)

})

output$table <- renderTable({

uploaded\_file()[, c(input$x\_axis, input$y\_axis, input$fac), drop = FALSE]

})

output$plot <- renderPlotly({

ggplot(uploaded\_file(), aes\_string(x = input$x\_axis, y = input$y\_axis)) +

geom\_point(aes\_string(col = input$fac)) +

scale\_color\_gradientn(colours = rainbow(5))+

ggtitle("") +

theme\_minimal()

})

}

# Run the application

shinyApp(ui = ui, server = server)