Shiny App Workshop (I)

ISSS 616 Applied Statistical Analysis with R (ASAR)

August 2022

SMU School of Computing and Information Systems

Masters of IT in Business AY 2022-2023 – Term 1

Concept + Introduction

Create app directory and file

Shiny app architecture

Your first Shiny app

Adding UI control

Adding behavior in server

Basic UI

Inputs

Outputs

Layout Design

Wireframing

layouts

Debugging

Reading tracebacks

Error handling

1. Concept + Introduction

Concept

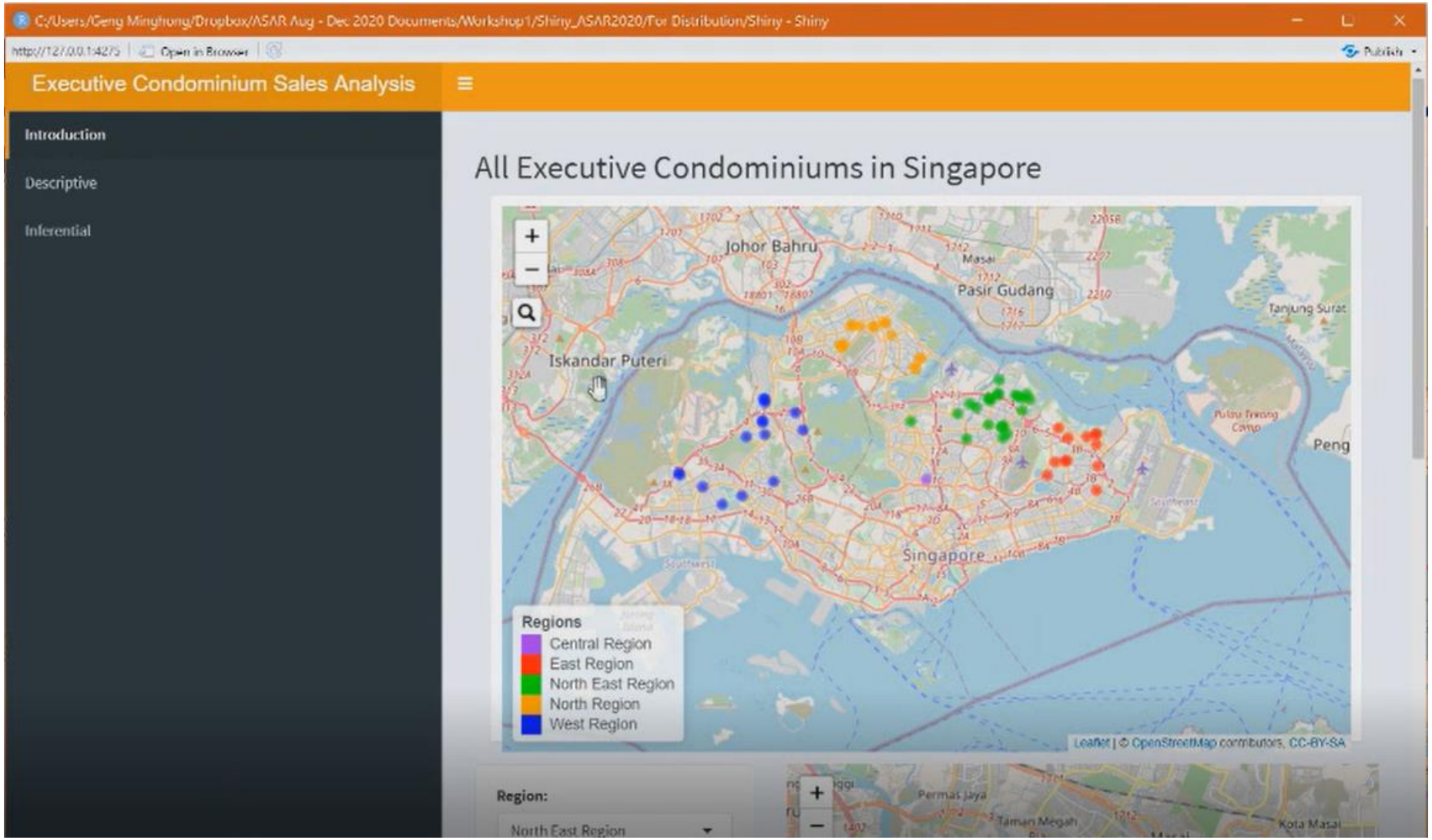
1st code

Basic UI

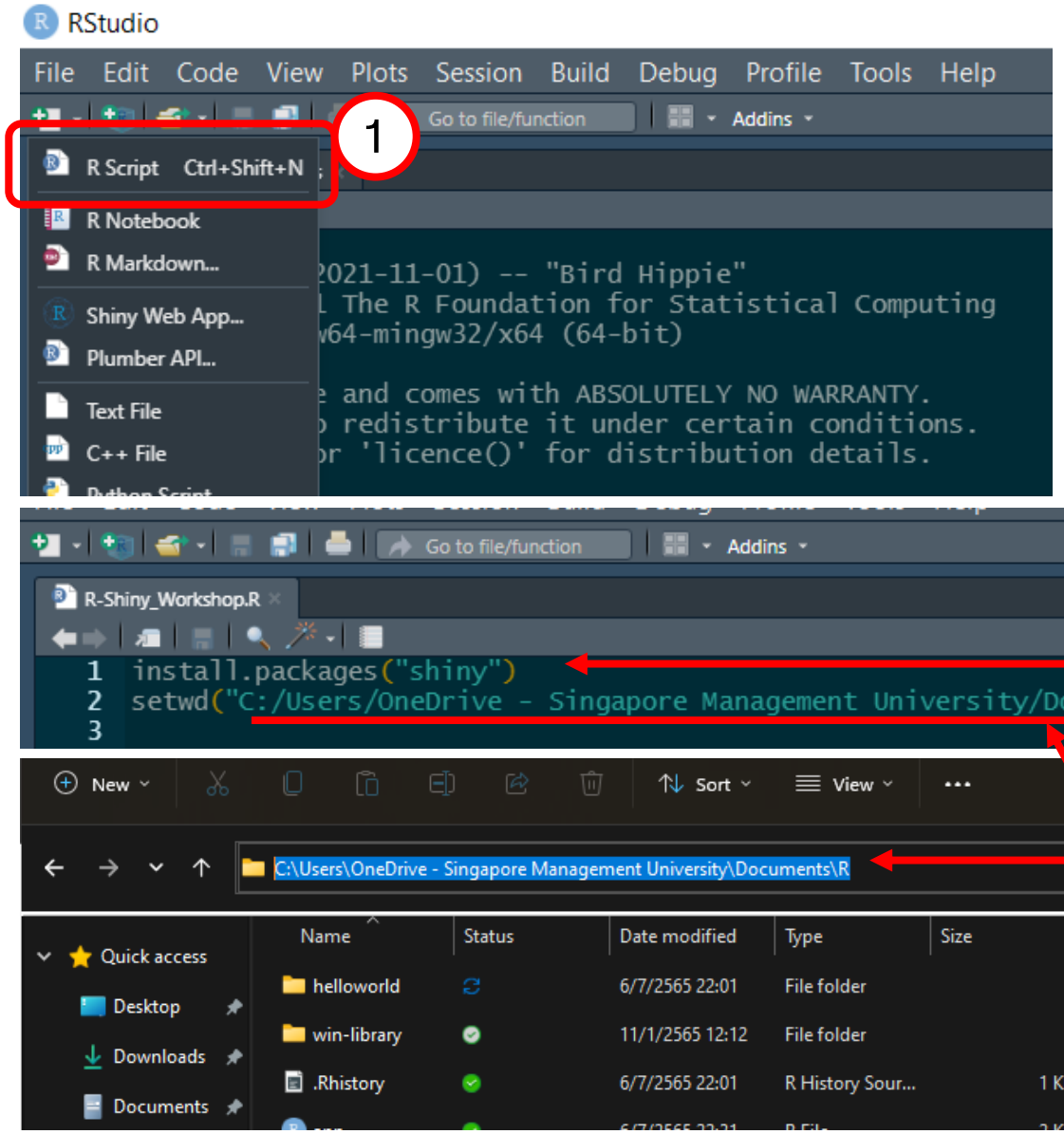
Layout

Debugging

Showcase



Create app directory and file



1. Create new R Script
2. Install Shiny package
3. Set up working directory

- Copy the path where you want to use as directory
- Paste the path in the command setwd
- Change \ to /

Create app directory and file

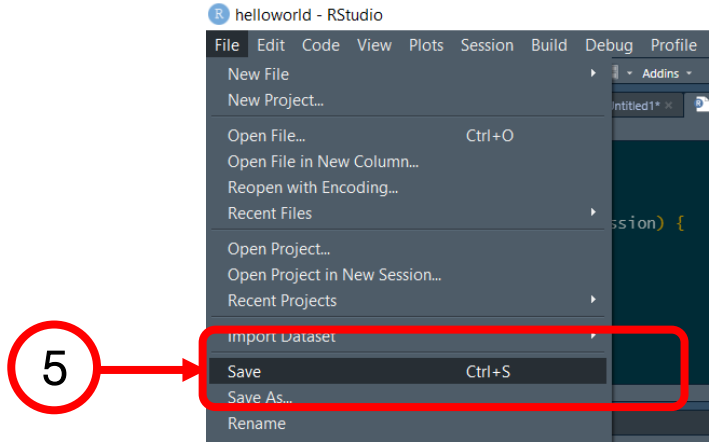

```
1
2
3  library(shiny)
4
5  ui <- fluidPage(
6    "Hello, world!"
7  )
8
9  server <- function(input, output, session) {
10 }
11
12 shinyApp(ui, server)
13
```

4

UI control

Server

4. Type following code
5. Save the file



This app.R does four things:

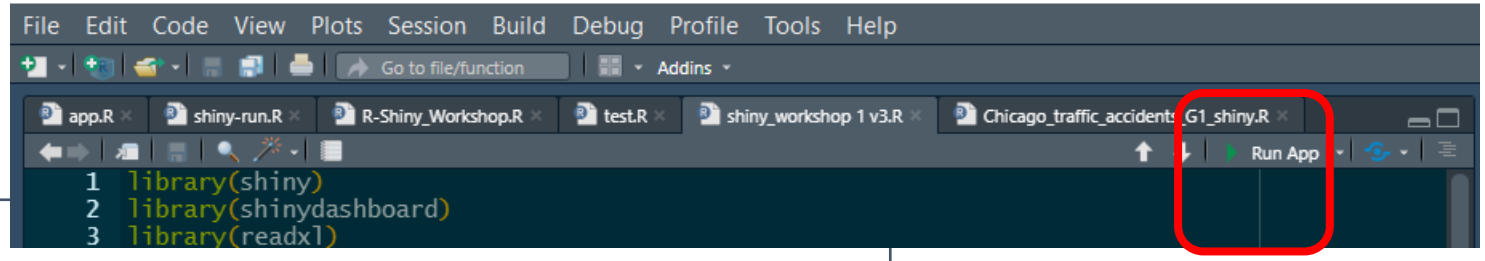
1. It calls library(shiny) to load the shiny package.

2. It defines the user interface, the HTML webpage that humans interact with. In this case, it's a page containing the words "Hello, world!".

3. It specifies the behavior of our app by defining a server function. It's currently empty, so our app doesn't do anything, but we'll be back to revisit this shortly.

4. It executes shinyApp(ui, server) to construct and start a Shiny application from UI and server.

Running and Stopping



Running

There are a few ways you can run this app:

1. Click the Run App button in the document toolbar.

2. Use a keyboard shortcut: Ctrl + Shift + Enter.

Stopping

Notice that R prompt isn't visible, and the console toolbar displays a stop sign. While a Shiny app is running, it "blocks" the R console means that you can't run new commands at the R console until the Shiny app stops.

There are a few ways you can stop this app:

1. Click the stop sign icon on the R console toolbar.

2. Close the Shiny app window.

The Shiny App architecture

Concept

1st code

Basic UI

Layout

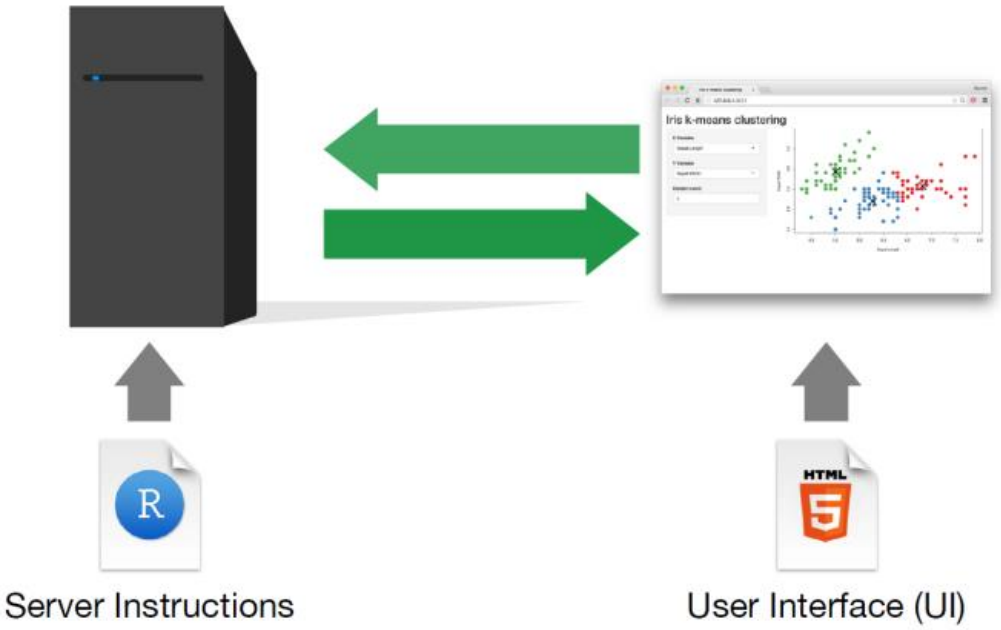
Debugging

```
1
2
3  library(shiny)
4
5  ui <- fluidPage(
6    "Hello, world!"
7  )
8
9  server <- function(input, output, session) {
10 }
11
12 shinyApp(ui, server)
```

UI

Server

Basic structure of every Shiny App



Server

- The Back-end of your app
- Where all the formulas, the coding run

UI (User Interface)

- The HTML of Shiny
- Front-end
- Imagine building a webpage, how do you want to build your layout, what kind of items you want to put, in what position, what tabs
- What kind of buttons and interactivity?

2. Your first Shiny app

Adding UI control

Concept

1st code

Basic UI

Layout

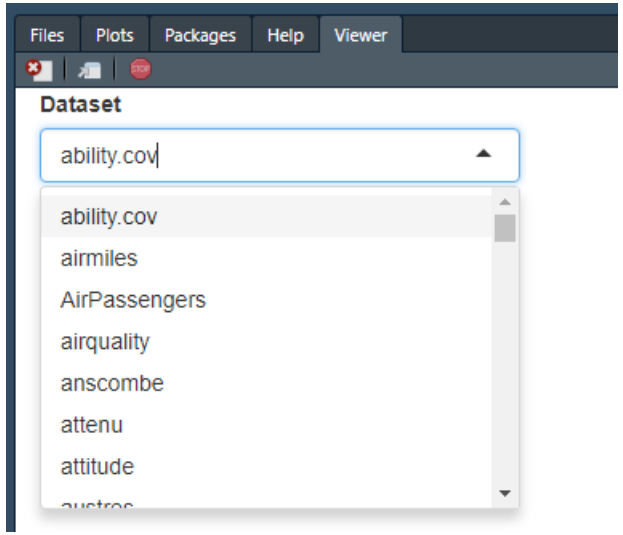
Debugging

```
4
5  library(shiny)
6  ui <- fluidPage(
7    selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),
8    verbatimTextOutput("summary"),
9    tableOutput("table")
10 )
```

Replacing UI with this code

we'll add some inputs and outputs to our UI.
Make a very simple app that shows you all the built-in dataset.

Result



selectInput() -- an input control that lets the user interact with the app by providing a value. In this case, it's a select box with the label "Dataset" and lets you choose one of the built-in datasets that come with R.

verbatimTextOutput() and **tableOutput() --** output controls, tell Shiny where to put rendered output.
verbatimTextOutput() displays code
tableOutput() displays tables

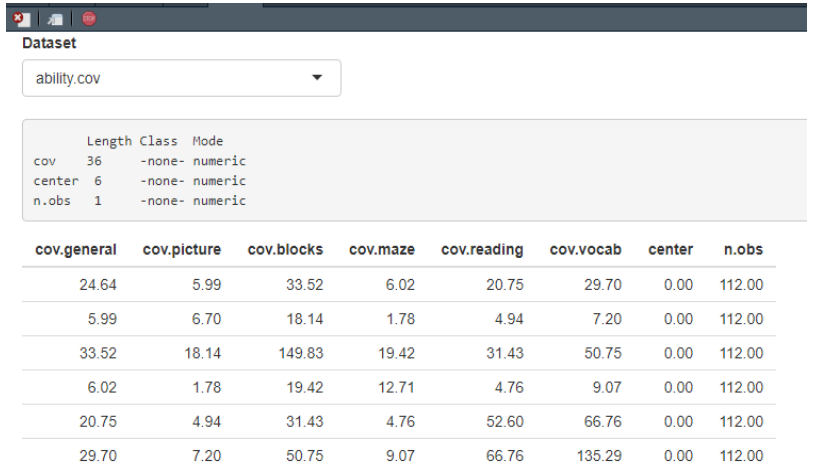
Adding behavior in server

It involves **telling Shiny how to perform a computation, not ordering Shiny to actually go do it**. It's like the difference between giving someone a recipe versus demanding that they go make you a sandwich.

We'll tell Shiny how to fill in the summary and table outputs in the sample app by providing the **"recipes"** for those outputs.


```
11 server <- function(input, output, session) {
12   output$summary <- renderPrint({
13     dataset <- get(input$dataset, "package:datasets")
14     summary(dataset)
15   })
16
17   output$table <- renderTable({
18     dataset <- get(input$dataset, "package:datasets")
19     dataset
20   })
21 }
```

Replacing empty server with this code



- The left-hand side of <-, output$**ID**, indicates that you're providing the recipe for the Shiny output with that **ID**.
- The right-hand side of <- uses a specific render function to wrap some code that you provide.

Learn from our first Shiny app

```
library(shiny)

ui <- fluidPage(
  selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),
  verbatimTextOutput("summary"),
  tableOutput("table")
)

server <- function(input, output, session) {
  output$summary <- renderPrint({
    dataset <- get(input$dataset, "package:datasets")
    summary(dataset)
  })

  output$table <- renderTable({
    dataset <- get(input$dataset, "package:datasets")
    dataset
  })
}

shinyApp(ui, server)
```

UI

Server

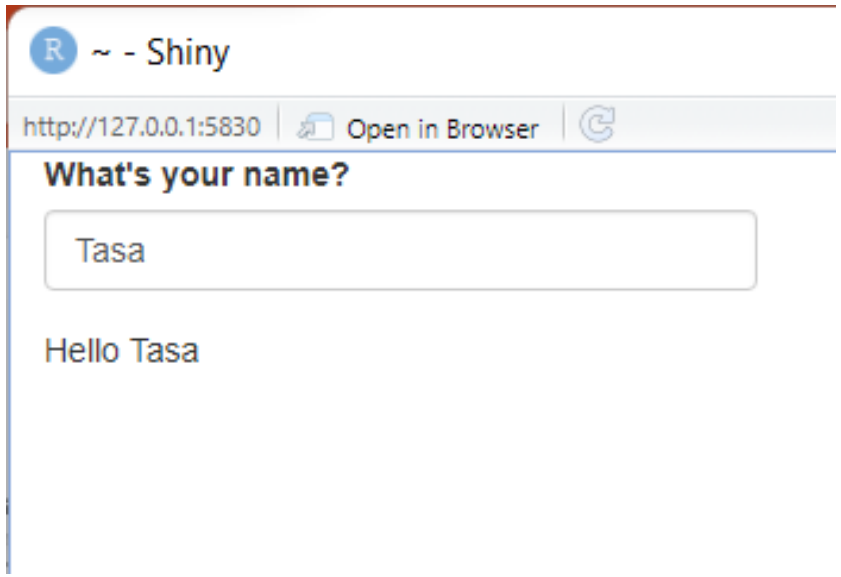
- Each render{Type} function is designed to produce different types of output (e.g. text, tables, and plots), and is often paired with a {type}Output function.
- For example, in this app,
- renderPrint() is paired with verbatimTextOutput() to display a statistical summary with fixed-width (verbatim) text
- renderTable() is paired with tableOutput() to show the input data in a table.

Concept Exercise

Create an app that greets the user by name. You don't know all the functions you need to do this yet, so I've included some lines of code below. Think about which lines you'll use and then copy and paste them into the right place in a Shiny app.

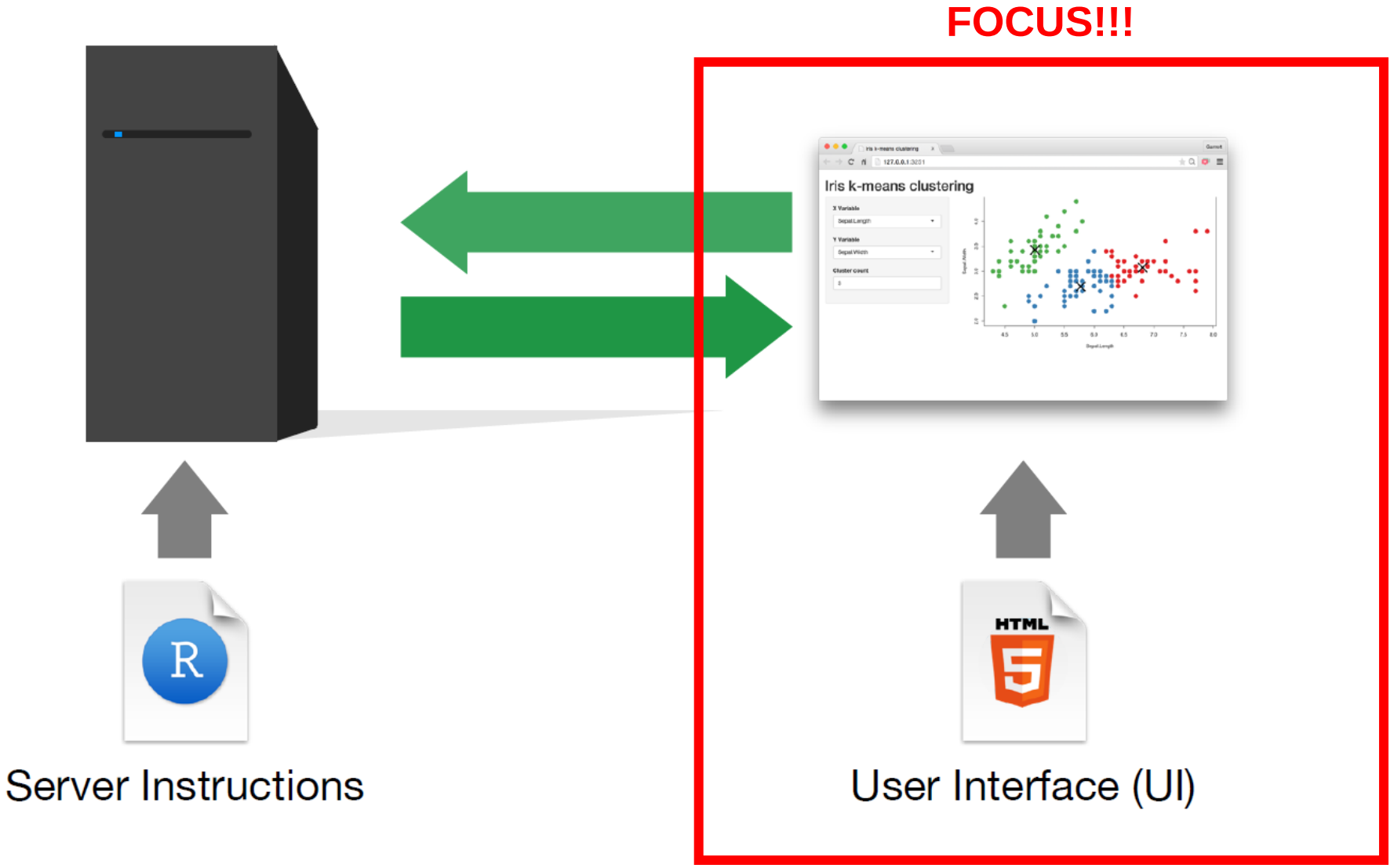
```
tableOutput("mortgage")
output$greeting <- renderText({
  paste0("Hello ", input$name)
})
numericInput("age", "How old are you?", value = NA)
textInput("name", "What's your name?")
textOutput("greeting")
output$histogram <- renderPlot({
  hist(rnorm(1000))
}, res = 96)
```

I want something just like this



3. Basic UI

Basic UI (User Interface)



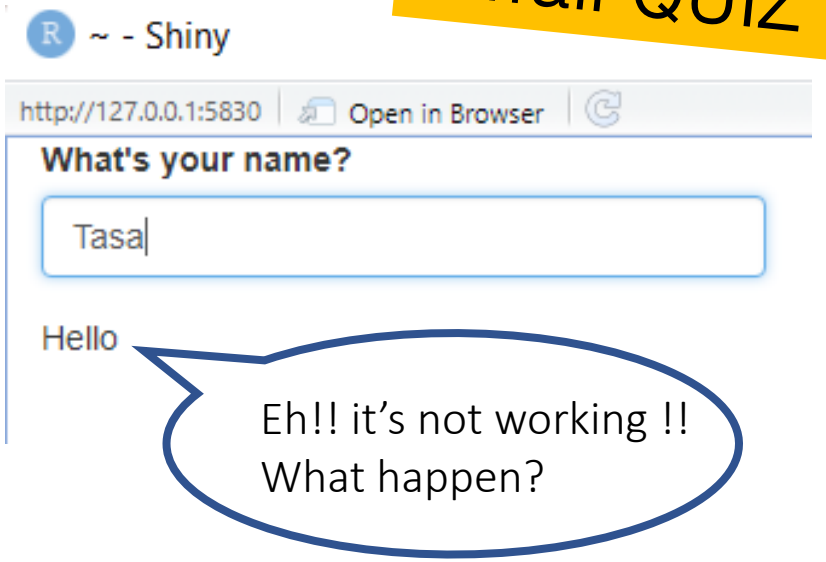
Inputs

ID

Label

```
2
3  library(shiny)
4  ui <- fluidPage(
5    textInput("names", "What's your name?"),
6    textOutput("greeting")
7  )
8
9  server <- function(input, output, session) {
10   output$greeting <- renderText({
11     paste0("Hello ", input$name)
12   })
13
14 }
15 shinyApp(ui, server)
```

Small QUIZ !!



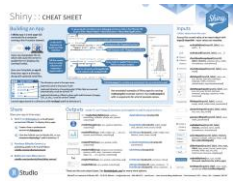
1. **inputID**
   Identifier used to connect the front end with the back end: if your UI has an input with ID, the server function will access it with that ID.

2. **Label** (Front-end)
   Create a human-readable label for the control.
   No restrictions on this string, but you'll need to make sure that your app is usable by humans!

Inputs widgets

Link for widget

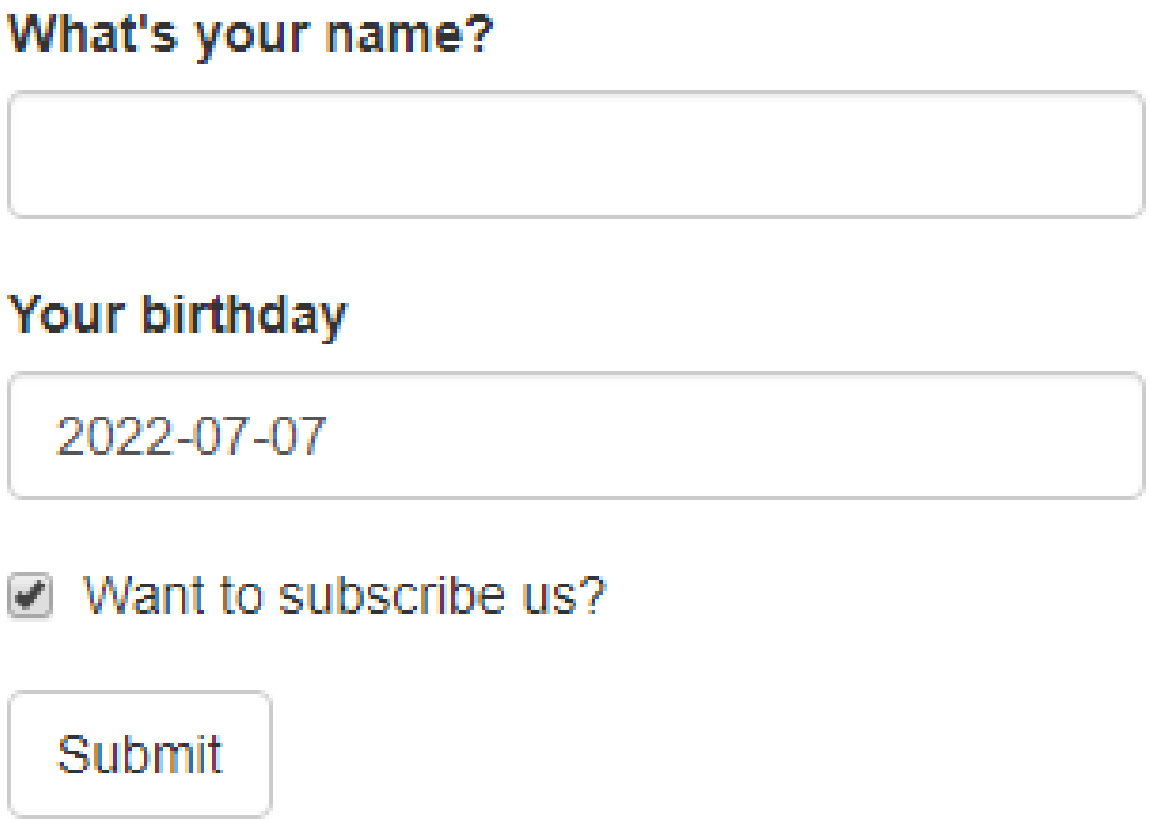
Cheat sheet



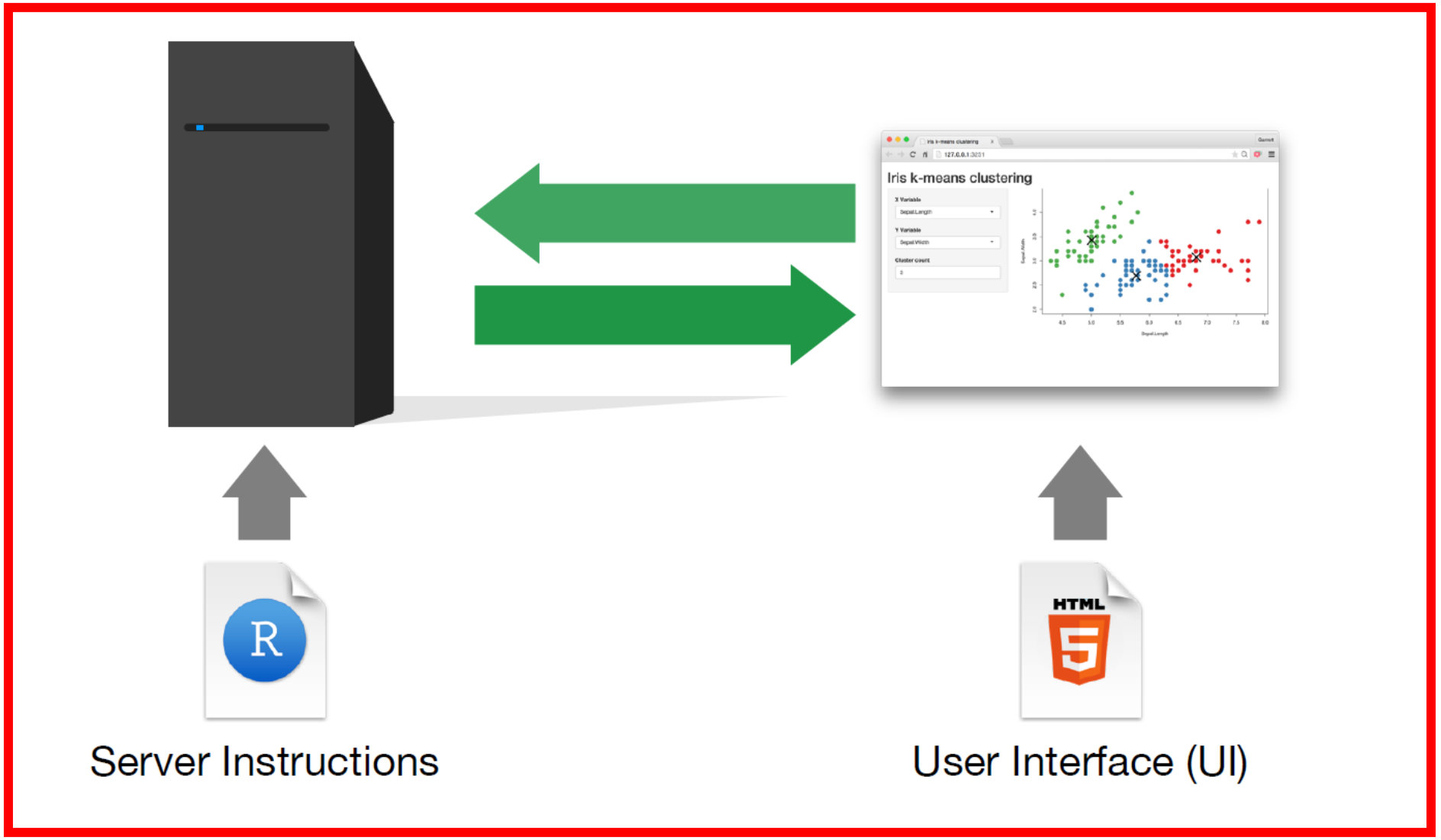
Action Submit	Choice A	2014-01-01	2017-06-21 to 2017-06-21
actionButton(inputId, label, icon, width, …)	**checkboxInput**(inputId, label, value, width)	**dateInput**(inputId, label, value, min, max, format, startview, weekstart, language, width, autoclose, datesdisabled, daysofweekdisabled)	**dateRangeInput**(inputId, label, start, end, min, max, format, startview, weekstart, language, separator, width, autoclose)
Browse... No file selected	1	Choice 1	Enter text...
fileInput(inputId, label, multiple, accept, width, buttonLabel, placeholder)	**numericInput**(inputId, label, value, min, max, step, width)	**selectInput**(inputId, label, choices, selected, multiple, selectize, width, size)	**textInput**(inputId, label, value, width, placeholder)
Choice 1 Choice 2 Choice 3	Choice 1 Choice 2 Choice 3	0 50 100 0 10 20 30 40 50 60 70 80 90 100	Choice 1 Choice 1 Choice 2
checkboxGroupInput(inputId, label, choices, selected, inline, width, choiceNames, choiceValues)	**radioButtons**(inputId, label, choices, selected, inline, width, choiceNames, choiceValues)	**sliderInput**(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post, timeFormat, timezone, dragRange)	**selectInput**(inputId, label, choices, selected, multiple, selectize, width, size)

UI Input Exercise

Let’s create an app to collect information



Basic UI (User Interface) and Server



Outputs

- If your UI specification creates an output with ID "plot", you'll access it in the server function with **output$plot**.
- Each output function on the front end is coupled with a render function in the back end.
- Each render{Type} function is often paired with a {type}Output function.

Matching command

Commands in UI	Commands in server
dataTableOutput()	DT::renderDataTable(
imageOutput()	renderImage()
plotOutput()	renderPlot()
verbatimTextOutput()	renderPrint()
tableOutput()	renderTable()
textOutput()	renderText()
uiOutput() htmlOutput()	RenderUI()
leafletOutput()	renderLeaflet()

```
library(shiny)

ui <- fluidPage(
  selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),
  verbatimTextOutput("summary"),
  tableOutput("table")
)

server <- function(input, output, session) {
  output$summary <- renderPrint({
    dataset <- get(input$dataset, "package:datasets")
    summary(dataset)
  })

  output$table <- renderTable({
    dataset <- get(input$dataset, "package:datasets")
    dataset
  })
}

shinyApp(ui, server)
```

Outputs widgets

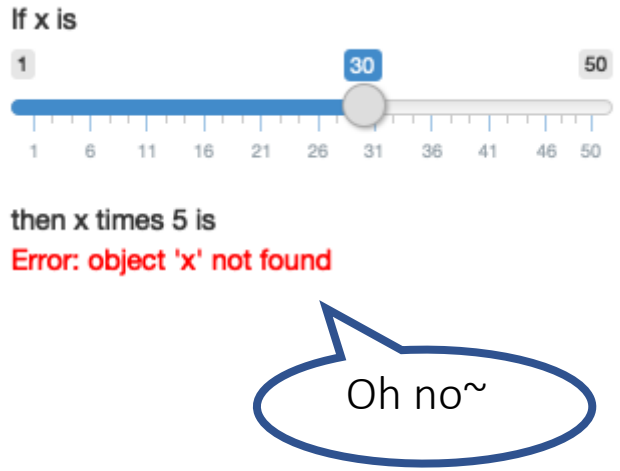
Cheat sheet

Commands in UI	Commands in server
DT::renderDataTable(expr, options, searchDelay, callback, escape, env, quoted, outputArgs**)**	**dataTableOutput(**outputId**)**
renderImage(expr, env, quoted, deleteFile, outputArgs**)**	**imageOutput(**outputId, width, height, click, dblclick, hover, brush, inline**)**
renderPlot(expr, width, height, res, …, alt, env, quoted, execOnResize, outputArgs**)**	**plotOutput(**outputId, width, height, click, dblclick, hover, brush, inline**)**
renderPrint(expr, env, quoted, width, outputArgs)	**verbatimTextOutput(**outputId, placeholder**)**
renderTable(expr, striped, hover, bordered, spacing, width, align, rownames, colnames, digits, na, …, env, quoted, outputArgs**)**	**tableOutput(**outputId**)**
renderText(expr, env, quoted, outputArgs, sep**)**	**textOutput(**outputId, container, inline**)**
renderUI(expr, env, quoted, outputArgs**)**	**uiOutput(**outputId, inline, container, …**)** **htmlOutput(**outputId, inline, container, …**)**

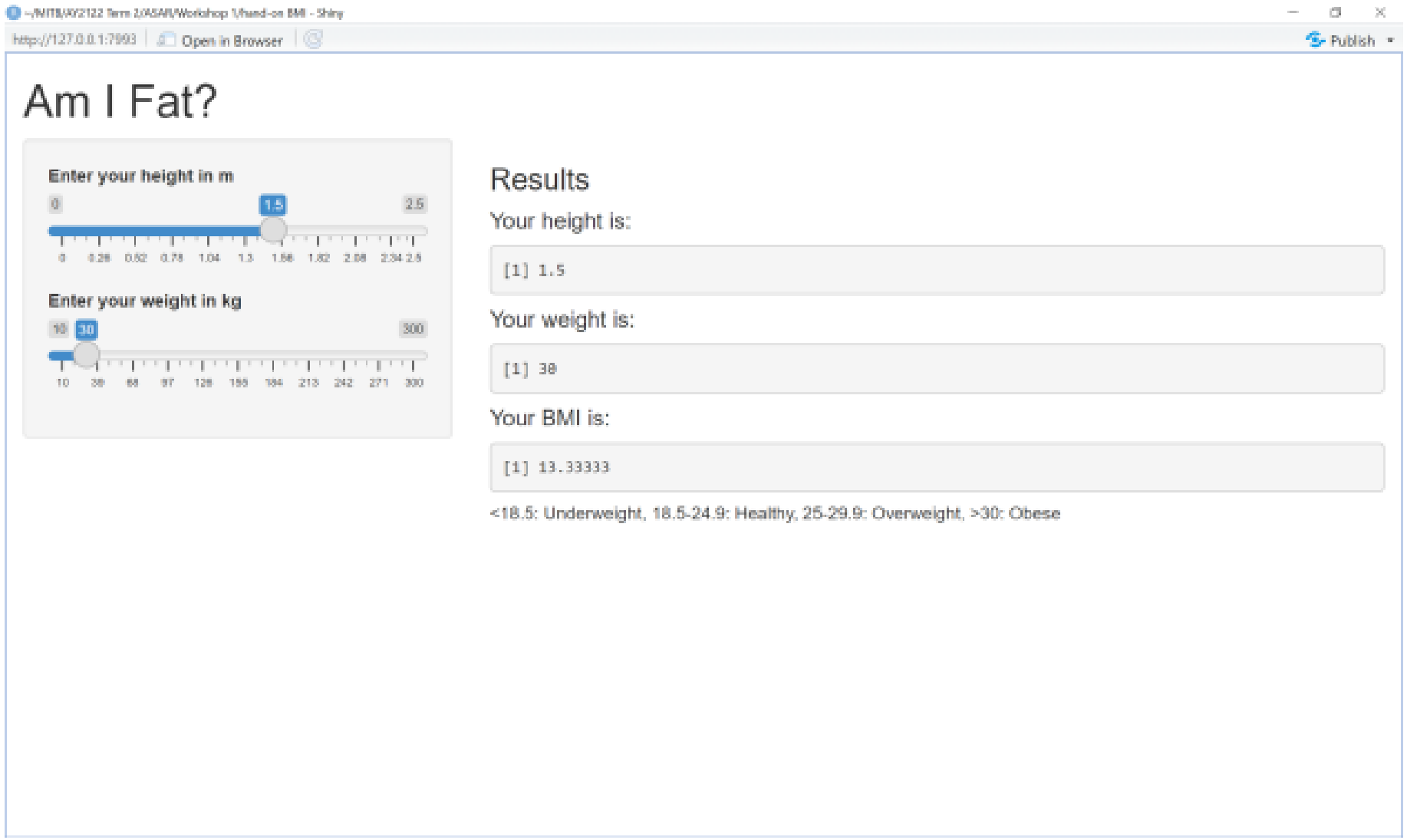
Server concept Exercise

Suppose your friend wants to design an app that allows the user to set a number (x) between 1 and 50, and displays the result of multiplying this number by 5. This is their first attempt: Unfortunately it has an error. Can you help them find and correct the error?

```
library(shiny)

ui <- fluidPage(
  sliderInput("x", label = "If x is", min = 1, max = 50, value = 30),
  "then x times 5 is",
  textOutput("product")
)

server <- function(input, output, session) {
  output$product <- renderText({
    x * 5
  })
}

shinyApp(ui, server)
```



Putting what we've learned together



Concept

1st code

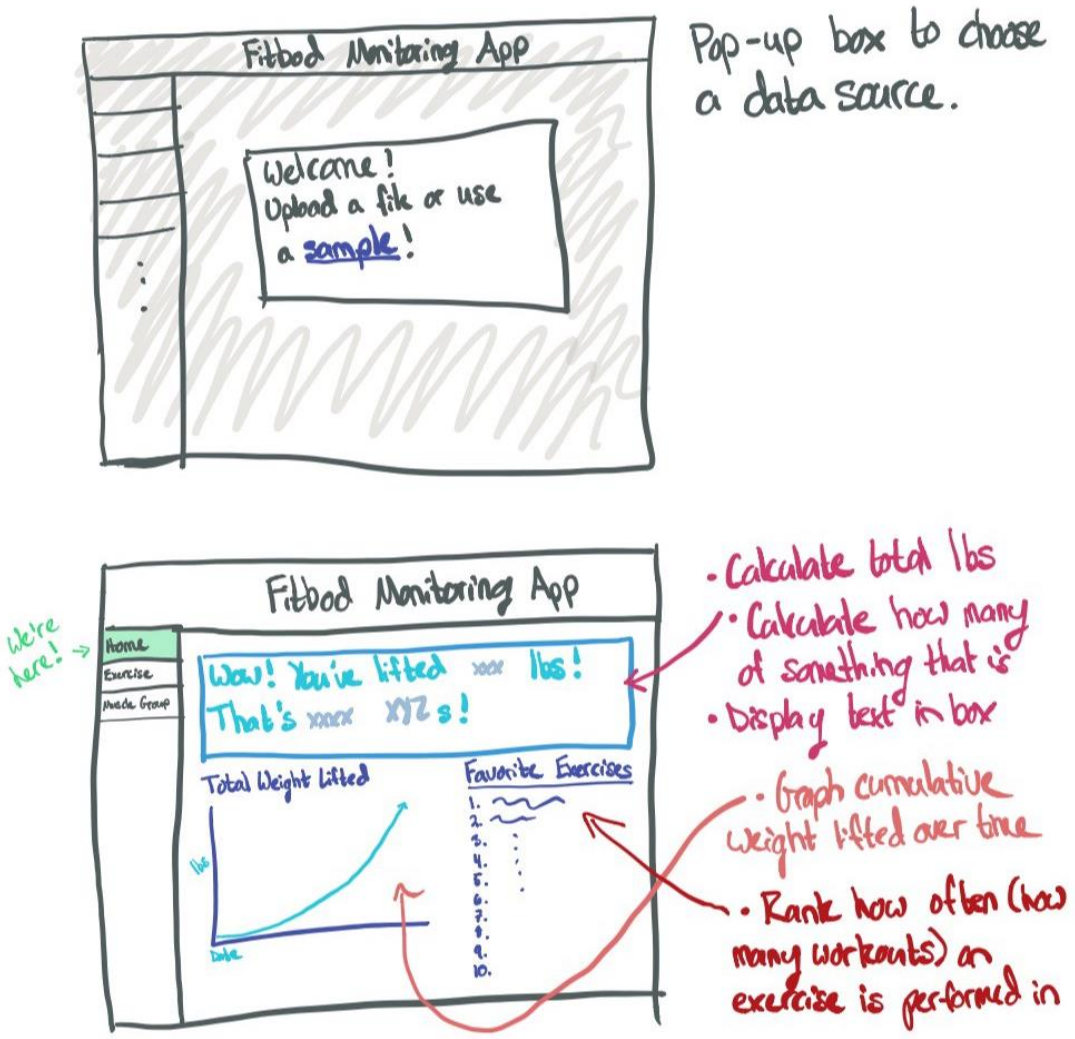
Basic UI

Layout

Debugging

4. Layout design

UI Wireframe



- Controlling the overall appearance of your app.
- In the 1st code, I didn't talk about how to lay them out on the page, and instead I just used fluidPage() to slap them together as quickly as possible.
- While this is fine for learning Shiny, it doesn't create usable or visually appealing apps, so now it's time to learn some more layout functions.

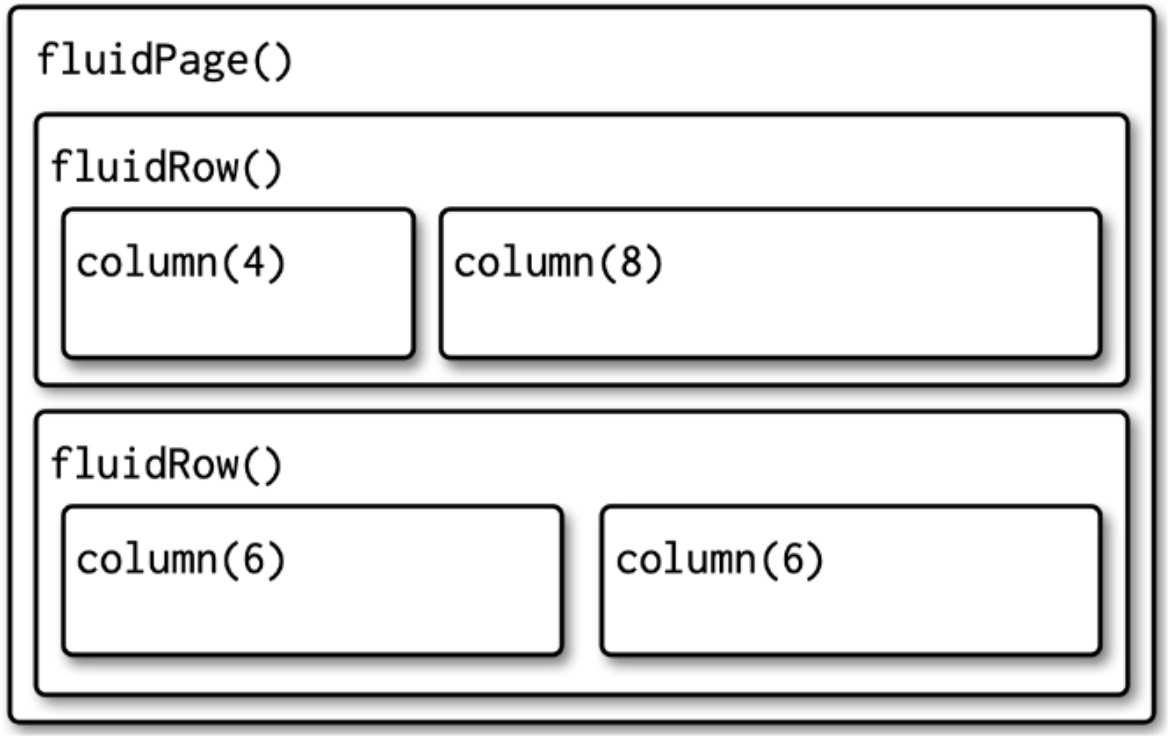
Picture from : https://towardsdatascience.com/production-grade-r-shiny-with-golem-prototyping-51b03f37c2a9

Layout with Shiny grid layout system

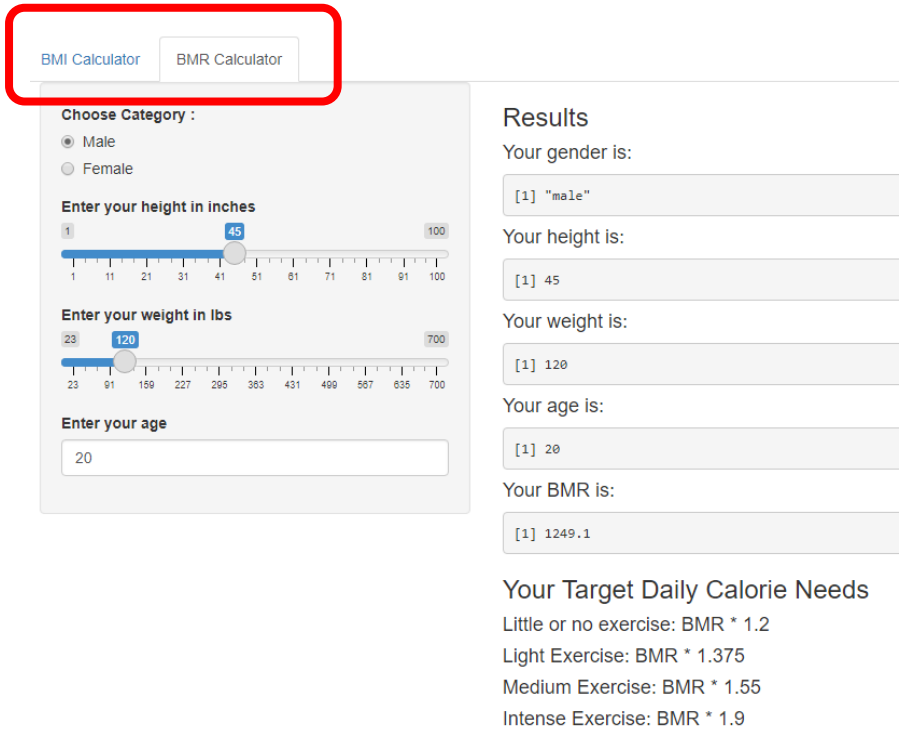
Layout functions provide the high-level visual structure of an app. Layouts are created by a hierarchy of function calls, where the hierarchy in R matches the hierarchy in the generated HTML.

In general, you start with fluidPage() as a big frame. The you use fluidRow() to create sections inside. And inside the fluidRow(), you can add columns with column().

```
fluidPage(
  fluidRow(
    column(4,
           ...
    ),
    column(8,
           ...
    )
  ),
  fluidRow(
    column(6,
           ...
    ),
    column(6,
           ...
    )
  )
)
```



Multi–page layout

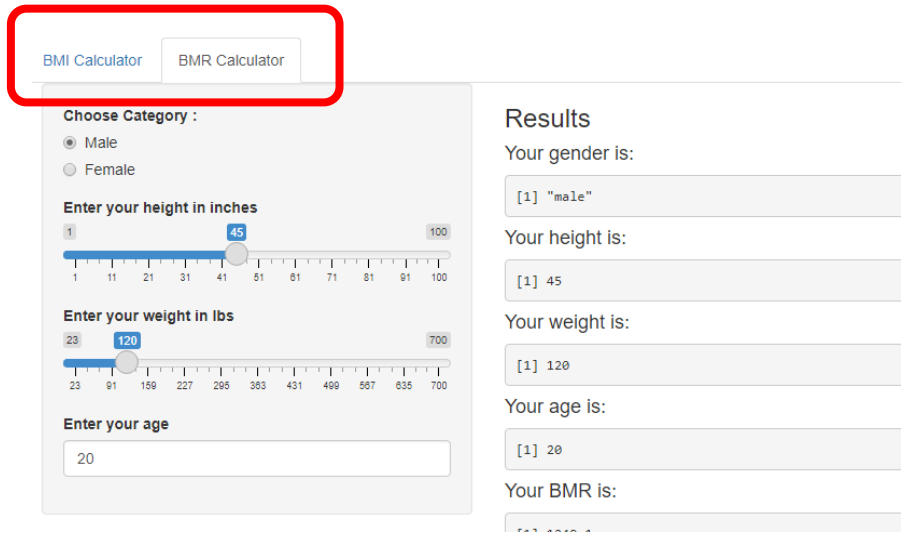



```
ui <- fluidPage(
   tabsetPanel(
      tabPanel("BMI calculator",
         ..
         ..
      )
      tabPanel("BMR calculator",
         ..
         ..
      )
)
```

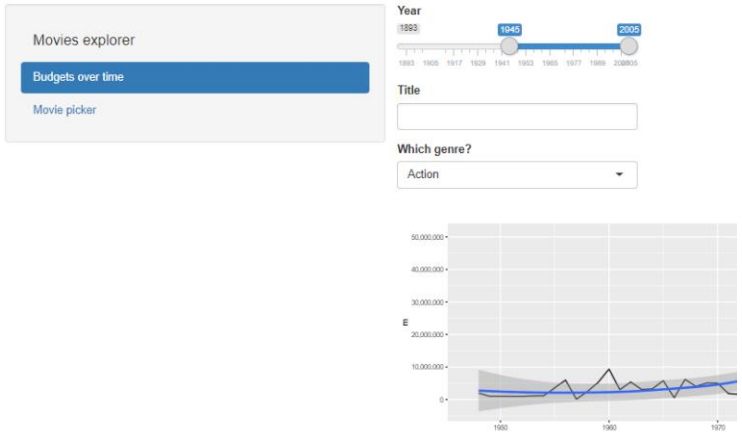
The simple way to break up a page into pieces is to use tabsetPanel() and tabPanel().

Multi–page layout

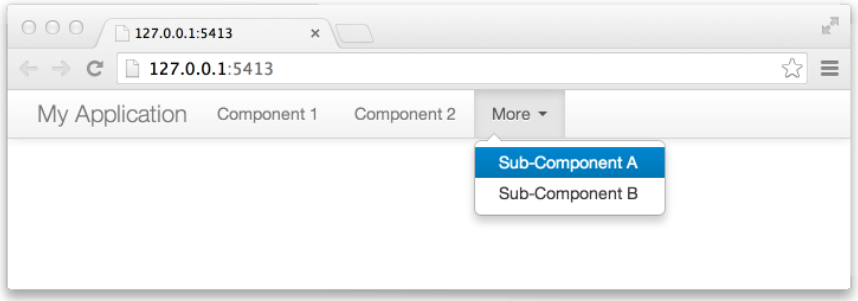
tabsetPanel() + tabPanel()



navlistPanel() + tabPanel()



navbarMenu() + tabPanel()



Picture from https://www.oreilly.com/library/view/hands-on-dashboard-development/9781789611557/a38b3024-01d9-4969-a1da-5422a3c7d124.xhtml

Putting what we've learned together

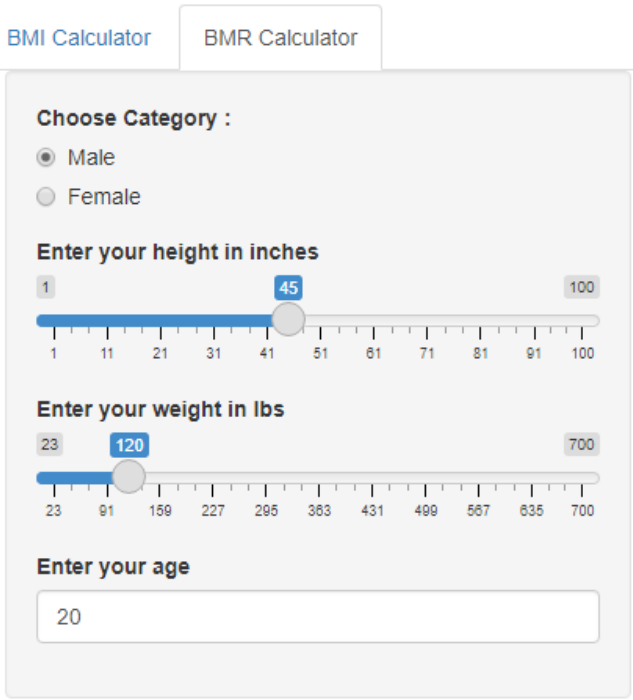
Concept

1st code

Basic UI

Layout

Debugging



Results

Your gender is:

```
[1] "male"
```

Your height is:

```
[1] 45
```

Your weight is:

```
[1] 120
```

Your age is:

```
[1] 20
```

Your BMR is:

```
[1] 1249.1
```

Your Target Daily Calorie Needs

Little or no exercise: BMR * 1.2

Light Exercise: BMR * 1.375

Medium Exercise: BMR * 1.55

Intense Exercise: BMR * 1.9

5. Debugging

1. Debugging

Pausing execution of your program, at a place you choose, to inspect its state as each following statement is executed. Best used when you suspect where a problem lies or need to verify the state around a particular section of code.

Solving before execution

2. Tracing

Collecting information as your program runs, without pausing it, for later analysis. Best used when you're diagnosing systemic issues (for instance, reactivity), when you can't debug, or when frequent interruption is inappropriate.

Finding the errors during execution.

3. Error handling

Finding the source of errors (both on the client and server side) and ascertaining their cause.

1. Debugging

Pausing execution of your program, at a place you choose, to inspect its state as each following statement is executed. Best used when you suspect where a problem lies or need to verify the state around a particular section of code.

Solving before execution

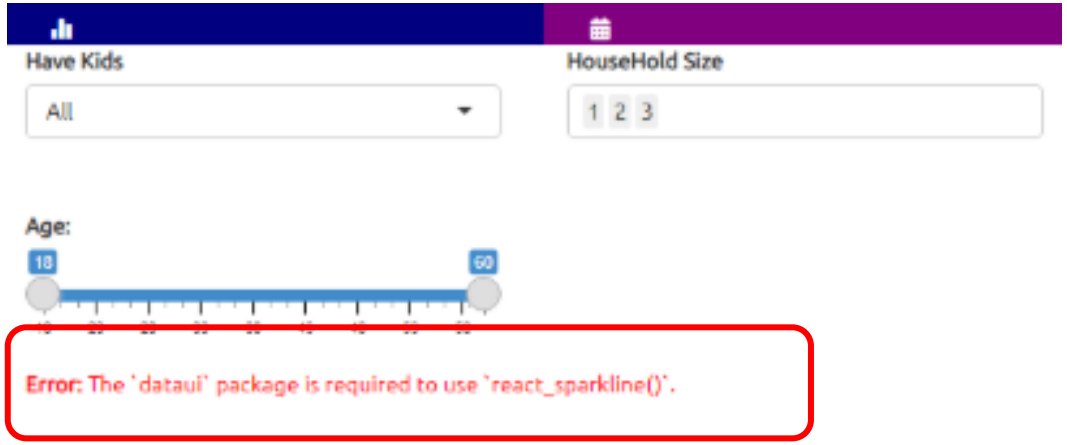
```
abl (4): participantia, nousenolasize, age, joviality
lgl (1): haveKids

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Error in readRDS(con, refhook = refhook) : error reading from connection
>
```

Such errors occur when we fail to import a file, or any syntax errors in the code. It doesn't provide output

2. Tracing

Collecting information as your program runs, without pausing it, for later analysis. Best used when you're diagnosing systemic issues (for instance, reactivity), when you can't debug, or when frequent interruption is inappropriate.

Finding the errors during execution.



Such errors allow the system to produce output yet the error appears in the specific place of graph.

3. Error handling

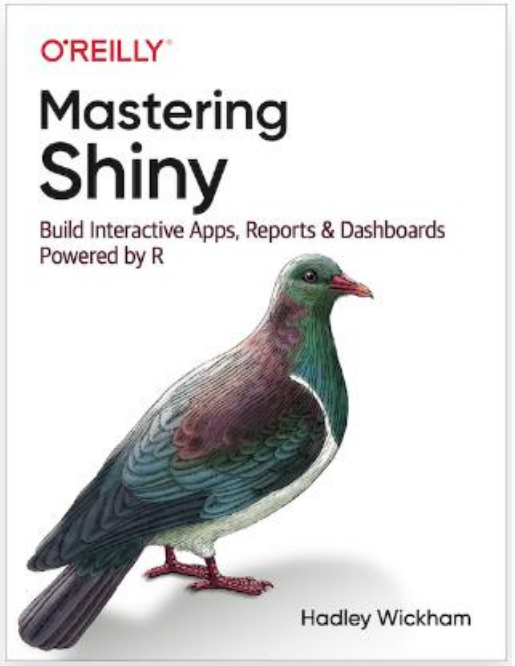
Finding the source of errors (both on the client and server side) and ascertaining their cause.

It doesn't produce any error. But the functionality may not be satisfied. It has to be verified by comparing ui and server functions.

Reference

Mastering Shiny, 2020, Hadley Wickham

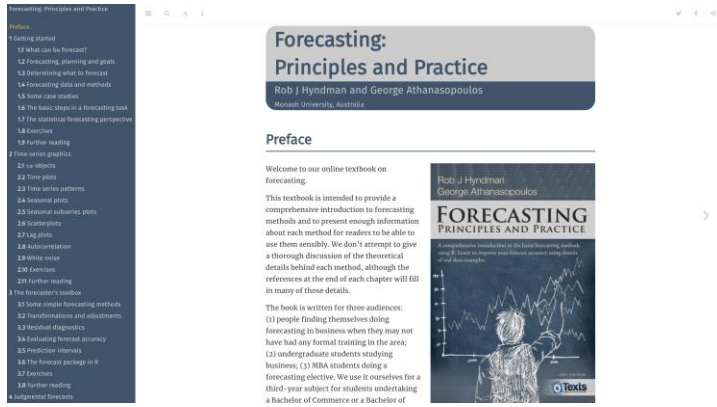
https://mastering-shiny.org/index.html



Resources for future dive in RShiny

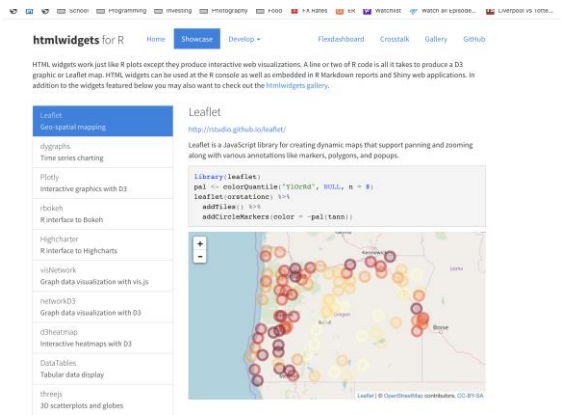
Shiny Website

https://shiny.rstudio.com/articles/basics.html



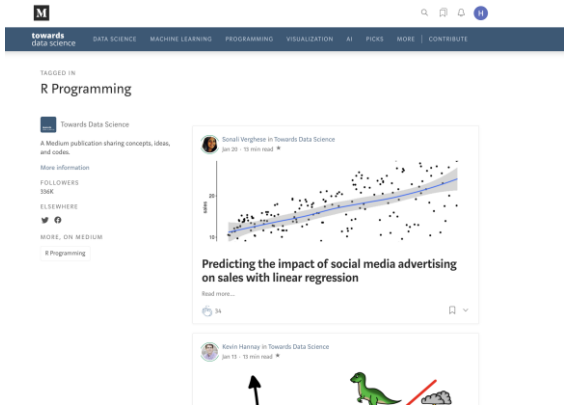
https://otexts.com/fpp2/

Widgets and libraries



http://www.htmlwidgets.org/showcase_leaflet.html

R Community



https://towardsdatascience.com/tagged/r-programming

Example for self-study

Loading of data

- Place the files within your working directory
- Call the following packages (install them if they have not been installed)
- Use the command read.csv (your working directory path) to run the files
- Pass them through the following:
  - EC Data – for ec_data_workshop.csv
  - Geo Location Coordinates – for ec_geo2.xlsx
  - Transaction Data – for ec_data_v6_mlr.csv

```
library(shiny)
library(shinydashboard)
library(readxl)
library(readr)
library(dplyr)
library(ggplot2)
library(leaflet)
library(leaflet.extras)

setwd("/Users/harrytsang/Dropbox/ASAR 2019-2020 Term 2/Workshops/2020 term 2 Workshop1")
ec_geo2 <- read_excel("ec_geo2.xlsx")
ec_data <- read_csv('ec_data_workshop.csv')
ec_data_mlr <- read_csv('ec_data_v6_mlr.csv')
```

Dashboard header and sidebar

Dashboard header and sidebar

Code:

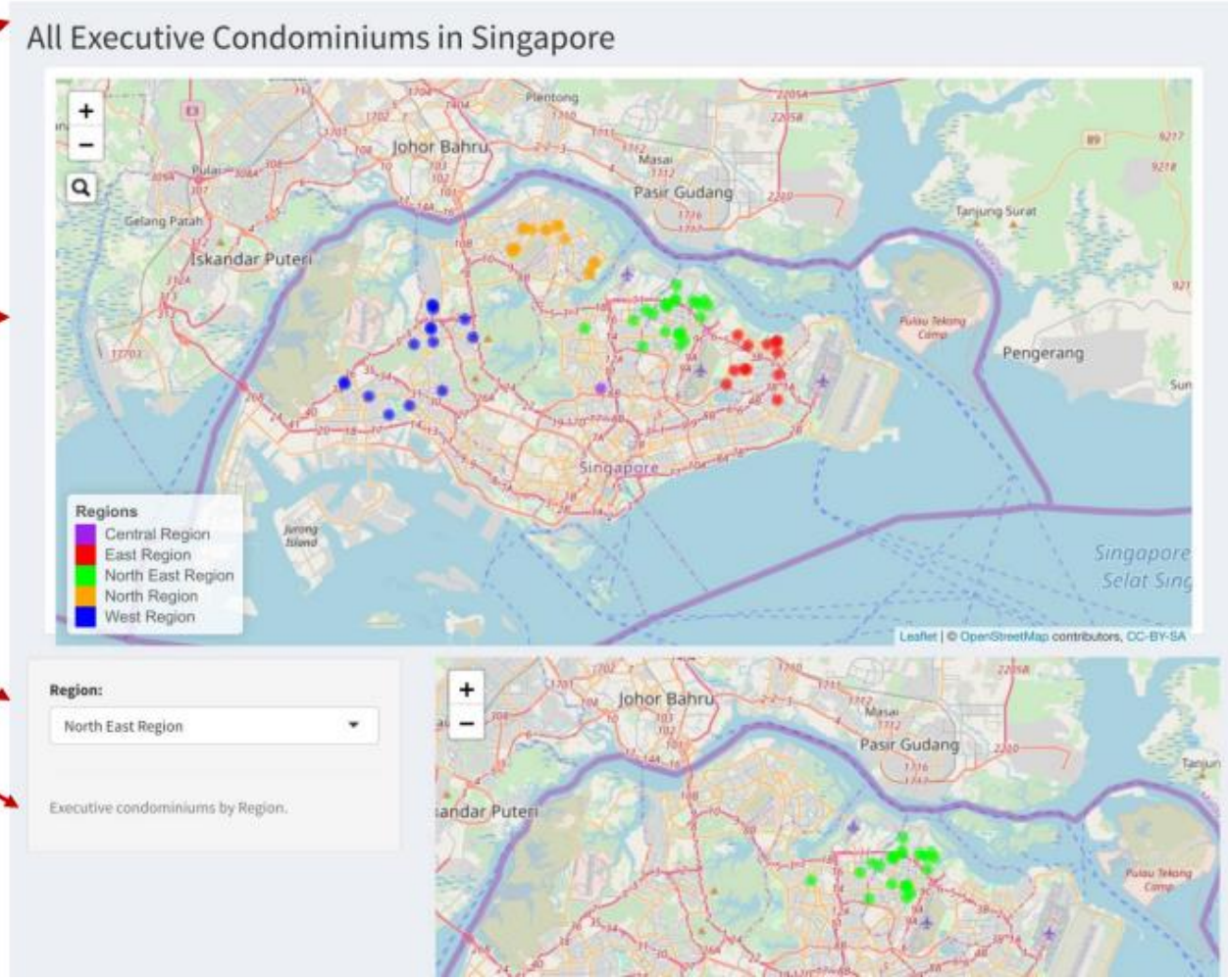
```
ui <- dashboardPage(skin = 'yellow',
                    dashboardHeader(title = 'Executive Condominium Sales Analysis', titleWidth = 400),
                    dashboardSidebar(width = 400,
                                     sidebarMenu(id = 'sbm',
                                                 menuItem('Introduction', tabName = 'Introduction', icon = NULL),
                                                 menuItem('Descriptive', tabName = 'Descriptive', icon = NULL),
                                                 menuItem('Inferential', tabName = 'Inferential', icon = NULL)
                                                 # menuItem('CI', tabName = 'CI', icon = NULL)
```

Output:

User Interface

Dashboard body

```
dashboardBody(
    tabItems(
        tabItem(tabName = 'Introduction',
                fluidPage(
                    titlePanel("All Executive Condominiums in Singapore"),
                    fluidRow(
                        column(width = 12,
                               box(
                                   width = 12,
                                   height = 500,
                                   solidHeader = TRUE,
                                   collapsible = FALSE,
                                   collapsed = FALSE,
                                   leafletOutput('map2', height = 500)
                               )
                        )
                    ),
                    sidebarLayout(
                        sidebarPanel(
                            selectInput("region", "Region:",
                                        choices =
                                            list('All' = list("North East Region", "North Region", "East Re
                                                              ))),


                            hr(),
                            helpText("Executive condominiums by Region."),
                            Position = "top"
                    ),
                        mainPanel(
                            leafletOutput("map", height = 500)
                        )
                    ),

                    fluidRow(
                        column(width = 12,
                               box(
                                   width = 12,
                                   height = 800,
                                   solidHeader = TRUE,
                                   collapsible = FALSE,
                                   collapsed = FALSE,
                                   plotOutput('regionanalysis', height = 750)
```



User Interface

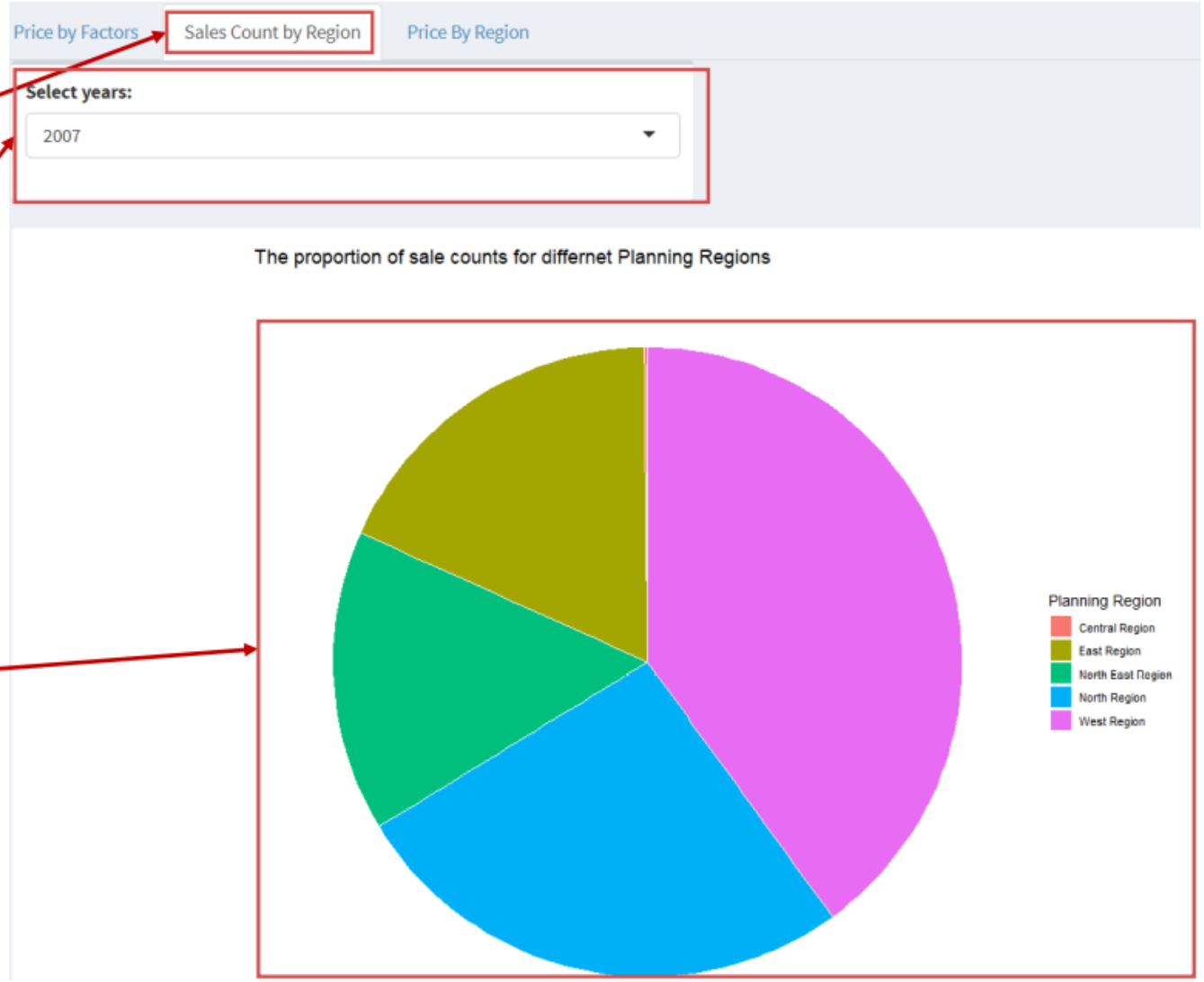
Dashboard body

```
(tabName = 'Descriptive',
 fluidPage(
     titlePanel("Price by Factors"),
     fluidRow(
         column(width = 12,
                box(
                    radioButtons('xcol',
                                 label = tags$strong('Analyse Sales By:'),
                                 choices = c('Floor level' = 'Floor_Level',
                                             'Distance from MRT' = 'mrt_distance_grp',
                                             'Completion date' = '`Completion Date`'),
                                 inline = TRUE)
                )) #end of box
     ),

     fluidRow(
         column(width = 12,
                box(
                    width = 12,
                    height = 800,
                    solidHeader = TRUE,
                    collapsible = FALSE,
                    collapsed = FALSE,
                    plotOutput('descriptiveAnalysis', height = 750)
                )
         )
```

User Interface

Dashboard body

```
tabPanel("Sales Count by Region",
         box(
             selectInput(
                 inputId = "Year",
                 label = "Select years:",
                 choices = 2000:2019
             )
         ),
         box(
             width = 12,
             height = 800,
             solidHeader = TRUE,
             collapsible = FALSE,
             collapsed = FALSE,
             plotOutput(outputId = "piechart",
                        height = "750"
             ))

),
```



User Interface

Dashboard body

```
tabPanel("Price By Region",
    width = 12,
    height = 800,
    solidHeader = TRUE,
    collapsible = FALSE,
    collapsed = FALSE,
    selectInput(
        inputId = "year_boxplot",
        label = "Select years:",
        choices = 2000:2019),
    plotOutput(outputId  = "boxplot",
               height = "600")
    ))
```

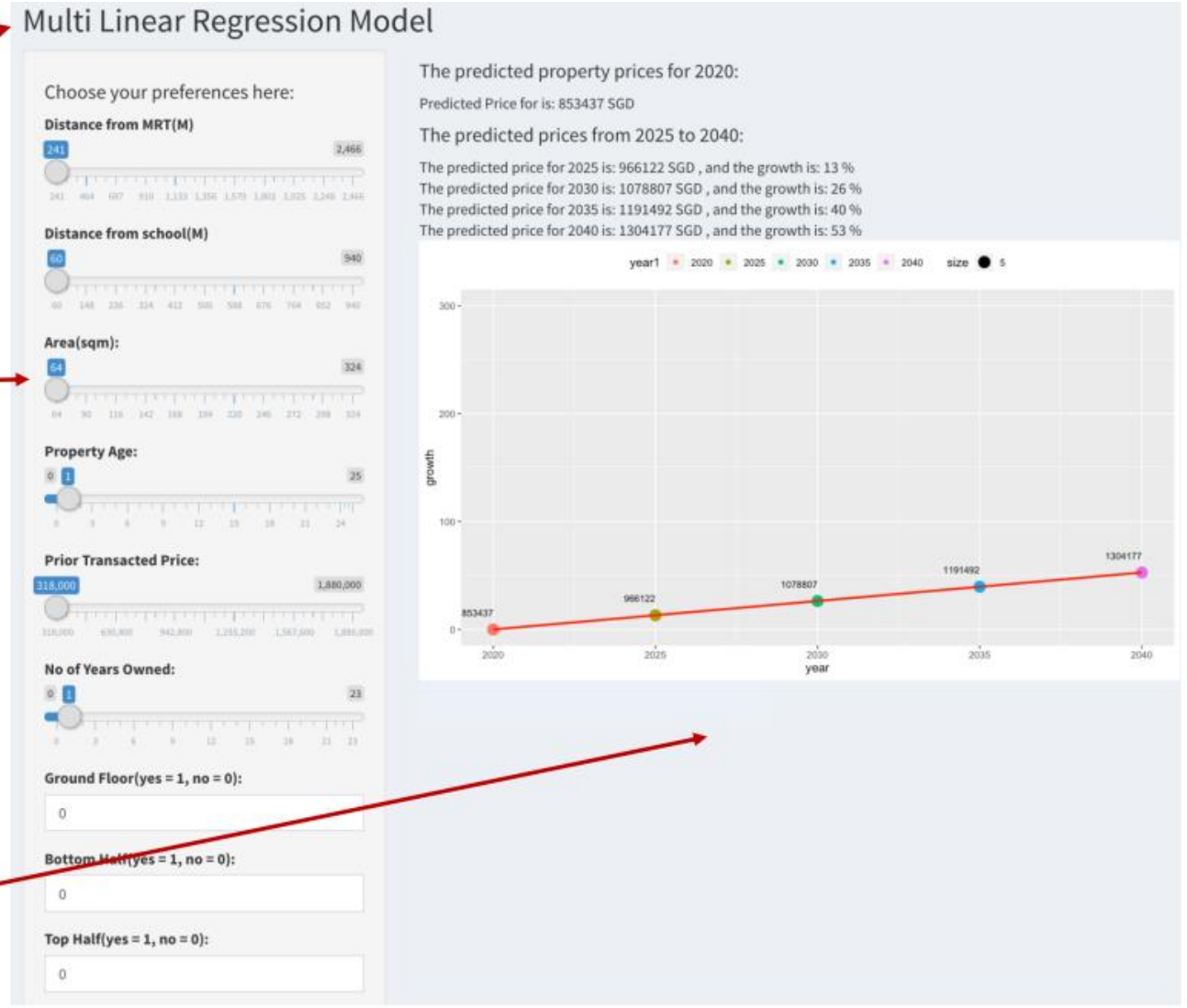
User Interface

Dashboard body

```
tabItem(tabName = 'Inferential',
        fluidPage(
            titlePanel('Multi Linear Regression Model')
            sidebarLayout(
                sidebarPanel(
                    h4("Choose your preferences here:"),

                    sliderInput("DMRT", "Distance from MRT(M)",
                                min = 241, max = 2466, value = 241
                    ),
                    sliderInput("DS", "Distance from school(M)",
                                min = 60, max = 940, value = 60
                    ),
                    sliderInput("Area", "Area(sqm):",
                                 min = 64, max = 324, value = 64
                                ),
                    sliderInput("PA", "Property Age:",
                                min = 0, max = 25, value = 1),
                    sliderInput("PTP", "Prior Transacted Price:",
                                min = 318000, max = 1880000, value = 318000),
                    sliderInput("NOYO", "No of Years Owned:",
                                min = 0, max = 23, value = 1),

                    numericInput("GF", "Ground Floor(yes = 1, no = 0):",
                                value = 0, min = 0, max = 1),
                    numericInput("BH", "Bottom Half(yes = 1, no = 0):",
                                 value = 0, min = 0, max = 1),
                    numericInput("TH", "Top Half(yes = 1, no = 0):",
                                 value = 0, min = 0, max = 1),
```

```
    mainPanel(
        h4("The predicted property prices for 2020:"),
        textOutput("formula"),
        h4("The predicted prices from 2025 to 2040:"),
        textOutput("price2025"),
        textOutput("price2030"),
        textOutput("price2035"),
        textOutput("price2040"),
        plotOutput("plot1"),
```

[No Title]



Shiny App UI & Server Integration

Leaflet

UI:

```
dashboardBody(
    tabItems(
        tabItem(tabName = 'Introduction',
                fluidPage(
                    titlePanel("All Executive Condominiums in Singapore"),
                    fluidRow(
                        column(width = 12,
                               box(
                                   width = 12,
                                   height = 500,
                                   solidHeader = TRUE,
                                   collapsible = FALSE,
                                   collapsed = FALSE,
                                   leafletOutput('map2', height = 500)
```

Server:

```
    output$map2 <- renderLeaflet({
        ec_geo2 %>%
            leaflet() %>%
            setView(lng = 103.8522, lat = 1.347510, zoom = 11) %>%
            addTiles() %>%
            addCircleMarkers(label = ~ pjname, color = ~ pal(`Planning Region`), radius = 3, fillOpacity = 0.5) %>%
            addSearchOSM() %>%
            addLegend(
                "bottomleft",
                pal = pal,
                values = ~`Planning Region`,
                opacity = 1, #color transparency
                title = "Regions")

    })
```

Notes:

1. If you name it map2, output as map2
2. When you using **leafletoutput** in UI, you have to **renderleaflet()** in server. Another Example will be **PlotOutput** + **RenderPlot** etc.
3. Fluid page, fluidrow are built-in functions in Shiny

Shiny App UI & Server Integration

ggplot

UI:

```
                radioButtons('xcol',
                             label = tags$strong('Analyse Sales By:'),
                             choices = c('Floor level' = 'Floor_Level',
                                         'Distance from MRT' = 'mrt_distance_grp',
                                         'Completion date' = '`Completion Date`'),
                             inline = TRUE)
            )) #end of box
    ),

    fluidRow(
        column(width = 12,
               box(
                   width = 12,
                   height = 800,
                   solidHeader = TRUE,
                   collapsible = FALSE,
                   collapsed = FALSE,
                   plotOutput('descriptiveAnalysis', height = 750)
               )
        )
```

Server:

```
output$descriptiveAnalysis <- renderPlot({
    analysis <- ec_data %>%
        group_by (.dots = input$xcol) %>%
        filter(`Completion Date` != 'Uncompleted', `Completion Date` != 'Unknown') %>%
        summarise(basket_value = mean(`Unit Price ($ psm)`, na.rm = T))

    p <- ggplot(analysis, aes_string(y = 'basket_value', x = input$xcol)) +
        geom_bar(aes_string(fill = input$xcol), stat = 'identity') +
        labs(title = 'Average Sales Price', subtitle = paste('by', input$xcol),
             x = input$xcol, y = 'Sales Price ($)',
             fill = input$xcol)
    return(p)

})
```

Notes:

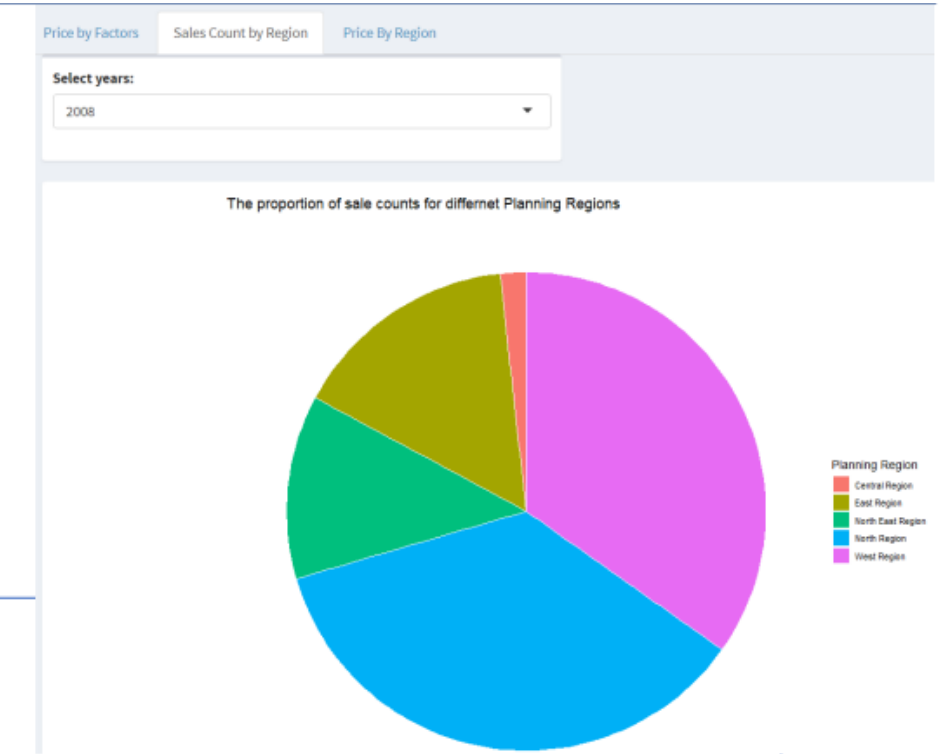
1. Group_by_ input, from radio buttons
2. %>% are pipe connectors from package **dplyr**, used to connect lines of codes to be neater.
3. ggplot Graphs in descriptive

Shiny App UI & Server Integration

ggplot

UI:

```
tabPanel("Sales Count by Region",
         box(
             selectInput(
                 inputId = "Year",
                 label = "Select years:",
                 choices = 2000:2019
             )
         ),
         box(
             width = 12,
             height = 800,
             solidHeader = TRUE,
             collapsible = FALSE,
             collapsed = FALSE,
             plotOutput(outputId = "piechart",
                        height = "800"
             ))
),
```



Server:

```
output$piechart <- renderPlot({
    ec_data %>%
        filter(Sale_Year == input$Year) %>%
        group_by( Planning Region ) %>%
        summarise(#region_value = mean(`Unit Price ($ psm)`, na.rm = T),
            count = n()) %>%
        ggplot(aes(x="", y=count, fill=`Planning Region`)) +
            geom_bar(stat="identity", width=1, color="white") +
            coord_polar("y", start=0)+
            theme_void() +
            labs(title = 'The proportion of sale counts for differnet Planning Regions',
                 x = 'Planning Region') +
            theme(plot.title = element_text(color = "Black", size = 14))
})
```

Shiny App UI & Server Integration

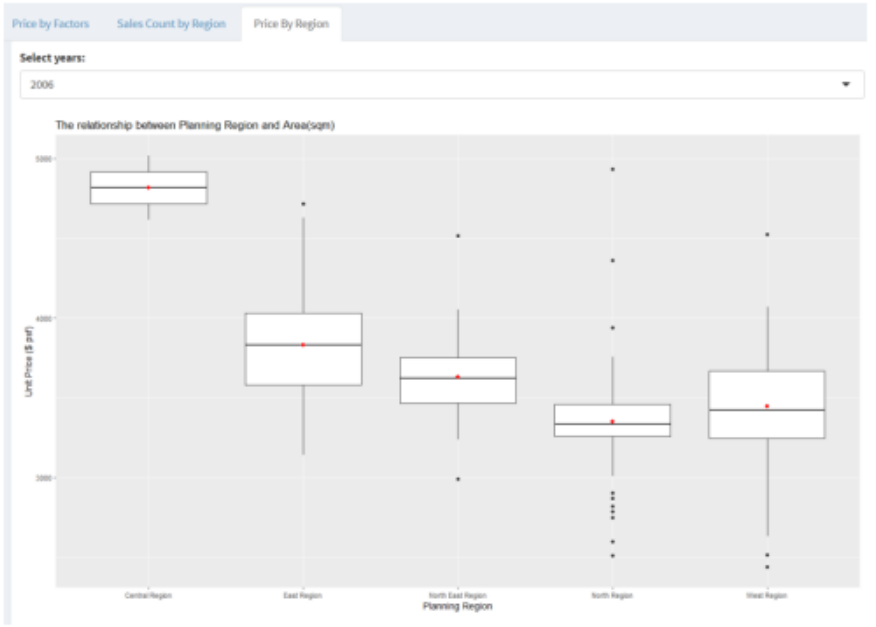
ggplot

UI:

```
tabPanel("Price By Region",
         width = 12,
         height = 800,
         solidHeader = TRUE,
         collapsible = FALSE,
         collapsed = FALSE,
         selectInput(
             inputId = "year_boxplot",
             label = "Select years:",
             choices = 2000:2019),
         plotOutput(outputId = "boxplot",
                    height = "600")
         ))
```

Server:

```
output$boxplot <- renderPlot({
    ec_data %>%
        filter(Sale_Year == input$year_boxplot) %>%
        ggplot(aes(y = `Unit Price ($ psm)`, x = `Planning Region`))+
            geom_boxplot()+
            stat_summary(geom = 'point',
                         fun = 'mean',
                         colour = 'red',
                         size = 2) +
            labs(title = 'The relationship between Planning Region and Area(sqm)',
                 x = 'Planning Region', y = 'Unit Price ($ psf)')
})
```



Data transformation in dplyr and RShiny

In dplyr

```
output$regionanalysis <- renderPlot({
    data_rv <- ec_data %>%
        group_by(`Planning Region`, Sale_Year) %>%
        summarise(region_value = mean(`Unit Price ($ psm)`, na.rm = T))
    data_rv$Sale_Year <- as.numeric(data_rv$Sale_Year)

    p <- ggplot(data_rv, aes(y = region_value, x = Sale_Year)) +
        geom_smooth(aes(col = data_rv$`Planning Region`), method = 'lm', se = FALSE) +
        geom_point(aes(col = data_rv$`Planning Region`), size = 2.5) +
        labs(title = 'Sales Price by Year', subtitle = paste('by', 'Region'),
             col = 'Planning Region',x = 'Sales Year', y = 'Sales Price ($)',
             fill = data_rv$`Planning Region`)
    return(p)

})
```

Data transformation in dplyr and RShiny

In RShiny(interactive)

```
output$descriptiveAnalysis <- renderPlot({
    analysis <- ec_data %>%
        group_by_(.dots = input$xcol) %>%
        filter(`Completion Date` != 'Uncompleted', `Completion Date` != 'Unknown') %>%
        summarise(basket_value = mean(`Unit Price ($ psm)`, na.rm = T))

    p <- ggplot(analysis, aes_string(y = 'basket_value', x = input$xcol)) +
        geom_bar(aes_string(fill = input$xcol), stat = 'identity') +
        labs(title = 'Average Sales Price', subtitle = paste('by', input$xcol),
             x = input$xcol, y = 'Sales Price ($)',
             fill = input$xcol)
    return(p)

    })
```