NATIONAL ENGINEERING CENTER
University of the Philippines
Diliman, Quezon City

6.0 Deploying R and Dashboard Generation

Eugene Rex L. Jalao, Ph.D.

Associate Professor

Department Industrial Engineering and Operations Research

University of the Philippines Diliman

@thephdataminer

Module 6 of the Business Intelligence and Analytics Certification of UP NEC and the UP Center for Business Intelligence

Outline for this Training

- Introduction to R and R Studio
- Data Types and Operators
  - Case Study on R Scripting
- Reading, Manipulating and Writing Data
  - Case Study on Dataset Analysis with ETL
- Basic R Programming
  - Case Study: Writing Functions
- Graphics and Plotting
- **Deploying R and Dashboard Generation**
  - **Case Study: Deploying a Simple Dashboard**
- Deploying R with C#
  - Case Study: A Simple Standalone GUI For R Apps

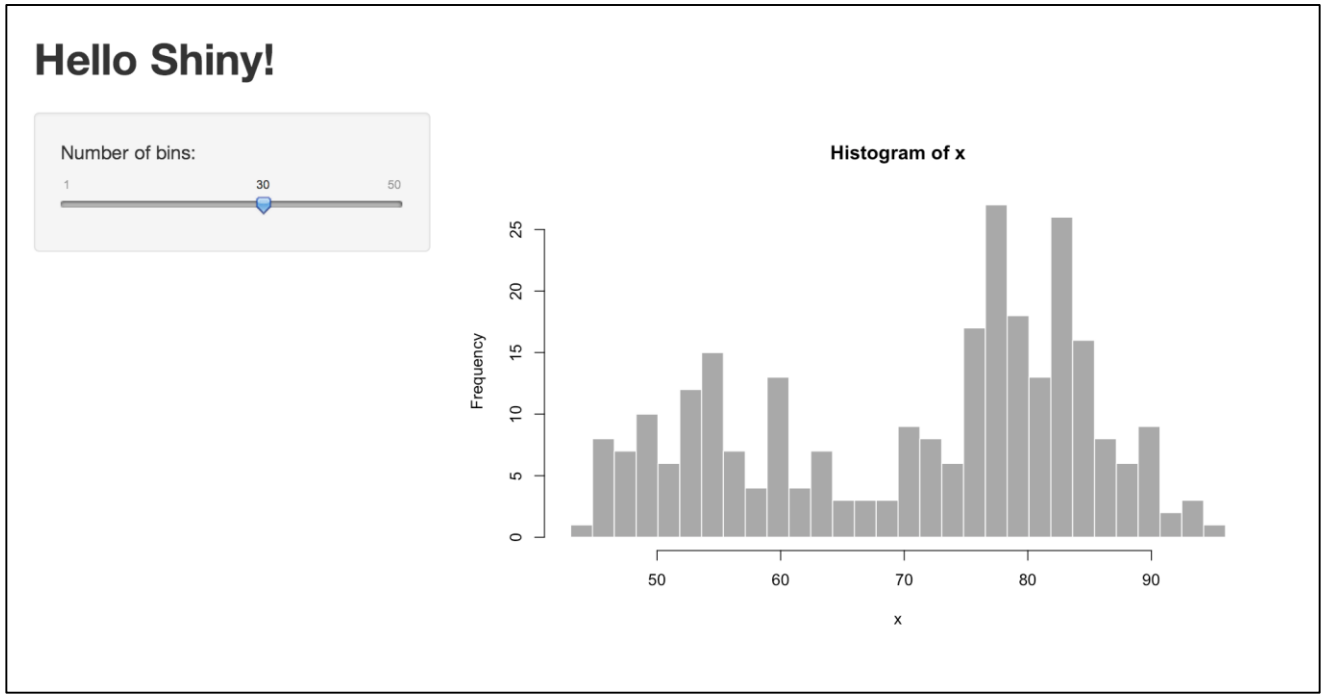
Outline for this Session

- The Shiny Package
- Building a Shiny App
- Reactive Output
- Shiny Execution Options

Shiny

Definition 6.1: Shiny

- Shiny is an R package that makes it easy to build **interactive web applications** (apps) straight from R.



Shiny

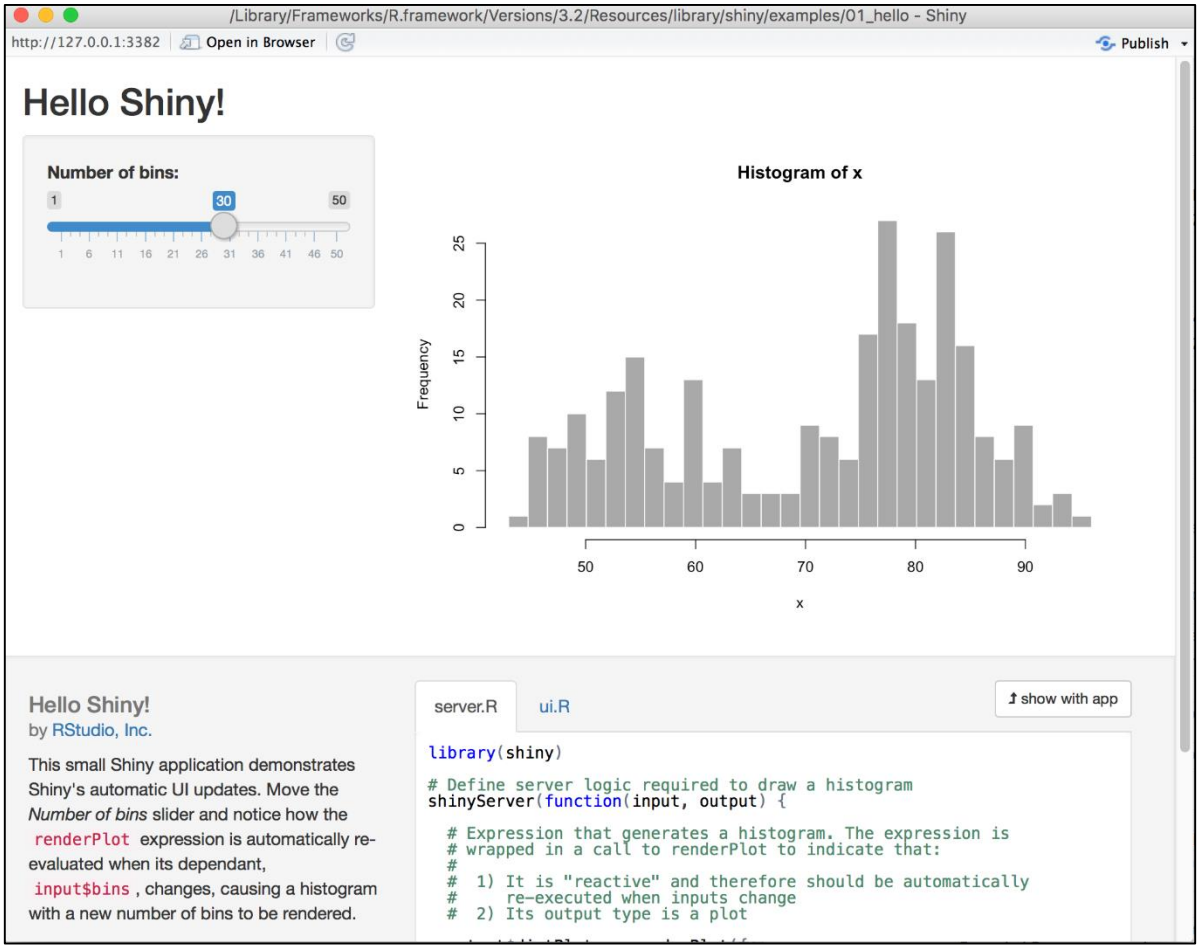
- The Shiny package has **eleven built-in examples** that each demonstrate how Shiny works.
- Each example is a self-contained Shiny app.
- The Hello Shiny example plots a histogram of R's **faithful dataset** with a configurable number of bins.
- Users can change the **number of bins** with a slider bar, and the app will immediately respond to their input.

Shiny

Example 6.1: Shiny

- `library(shiny)`
- `runExample("01_hello")`

Shiny

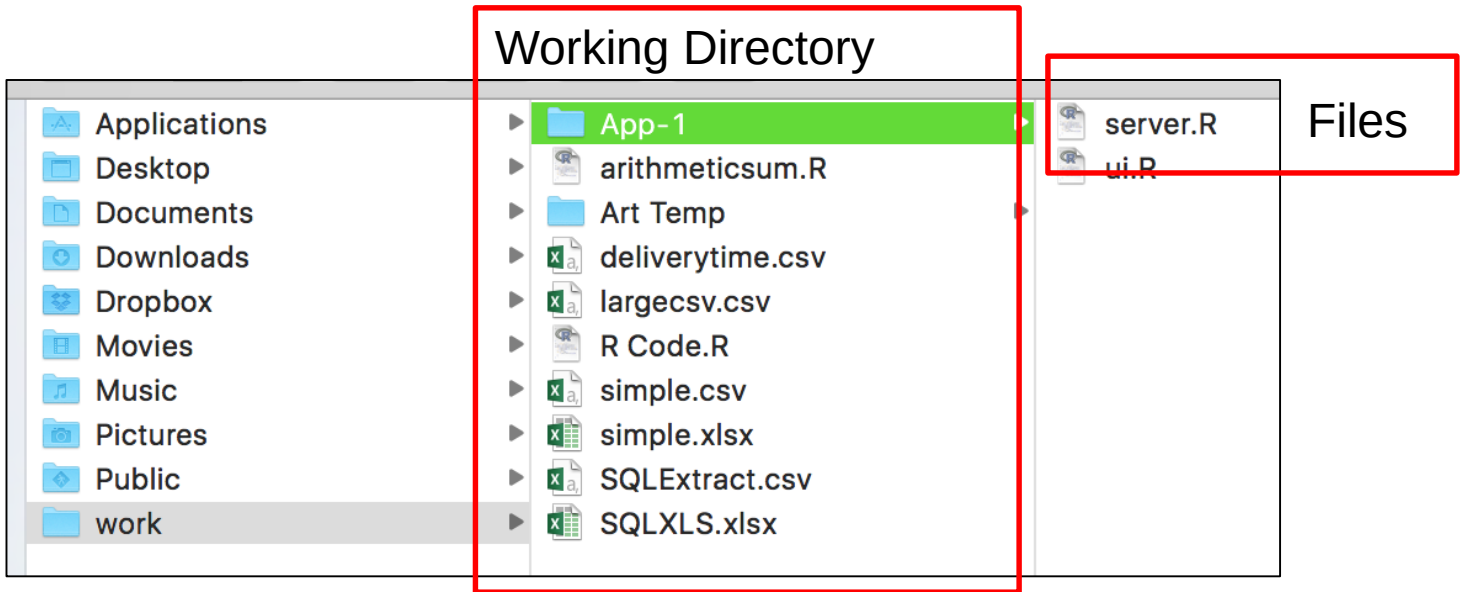

/Library/Frameworks/R.framework/Versions/3.2/Resources/library/shiny/examples/01_hello - Shiny
http://127.0.0.1:3382
Open in Browser
Publish
Hello Shiny!
Number of bins:
1
30
50
1 6 11 16 21 26 31 36 41 46 50
Histogram of x
Frequency
0 5 10 15 20 25
50 60 70 80 90
x
Hello Shiny!
by RStudio, Inc.
This small Shiny application demonstrates Shiny's automatic UI updates. Move the Number of bins slider and notice how the renderPlot expression is automatically re-evaluated when its dependant, input$bins , changes, causing a histogram with a new number of bins to be rendered.
server.R
ui.R
show with app
library(shiny)
# Define server logic required to draw a histogram
shinyServer(function(input, output) {
# Expression that generates a histogram. The expression is
# wrapped in a call to renderPlot to indicate that:
#
1) It is "reactive" and therefore should be automatically
re-executed when inputs change
2) Its output type is a plot

Shiny

- Shiny apps have **two components**:
- A **user-interface** script
  - The user-interface (ui) script controls the layout and appearance of your app. It is defined in a source script named ui.R.
- A **server script**
  - The server.R script contains the instructions and computations that the UI needs for display.

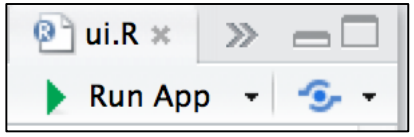
Shiny

- Starting a New Project
  - Every Shiny app has the same structure: **two R scripts saved** together in a directory. At a minimum, a Shiny app has **ui.R** and **server.R** files.
  - A Shiny app can be created by making a new directory and saving a ui.R and server.R file inside it.
  - Each app will need its own unique directory.



Shiny

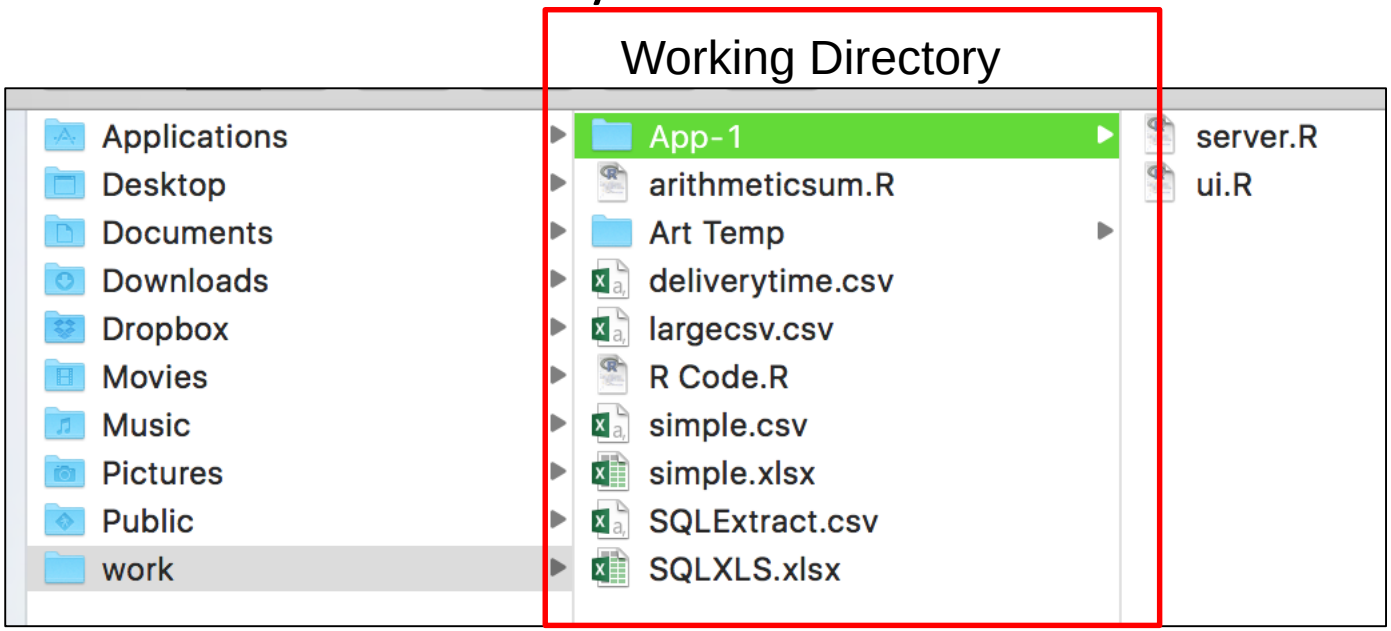
- Running a New Project
  - **Option 1**
    - A Shiny app can be run by giving the name of its directory to the function **runApp**. For example if your Shiny app is in a directory called my_app, run it with the following code:
    - `library(shiny)`
    - `runApp("my_app")`
  - **Option 2**
    - Open either ui.R or server.R
    - Click on Run App



Shiny

Example 6.2: Starting a New Package

- Create a new folder named **App-1** in your "work" folder.
- Two files ui.R and server.R needs to be written. When you are finished the directory should look like this:



Shiny

- Create a **new R script**. Name it server.R and save it inside the **App-1 Folder** and type the following code.

```
1  library(shiny)
2  shinyServer(function(input, output) {
3    output$distPlot <- renderPlot({
4      x    <- faithful[, 2]
5      bins <- seq(min(x), max(x),
6                  length.out = input$bins + 1)
7      hist(x, breaks = bins,
8           col = 'darkgray', border = 'white')
9    })
10 })
```

Shiny

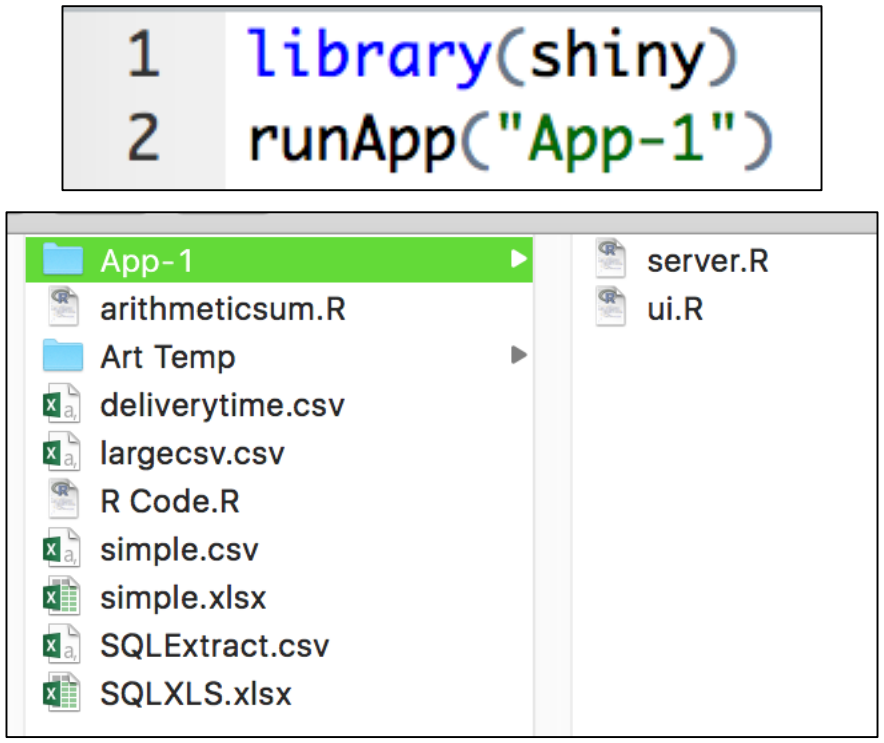
- Create a **new R script**. Name it ui.R and save it inside the **App-1 Folder** and type the following code.

```
1  library(shiny)
2  shinyUI(fluidPage(
3    titlePanel("Hello Rex!"),
4    sidebarLayout(
5      sidebarPanel(
6        sliderInput("bins","Number of bins:"
7                    ,min = 5, max = 50,value = 30)
8      ),
9      mainPanel(
10       plotOutput("distPlot")
11     )
12   )
13 ))
```

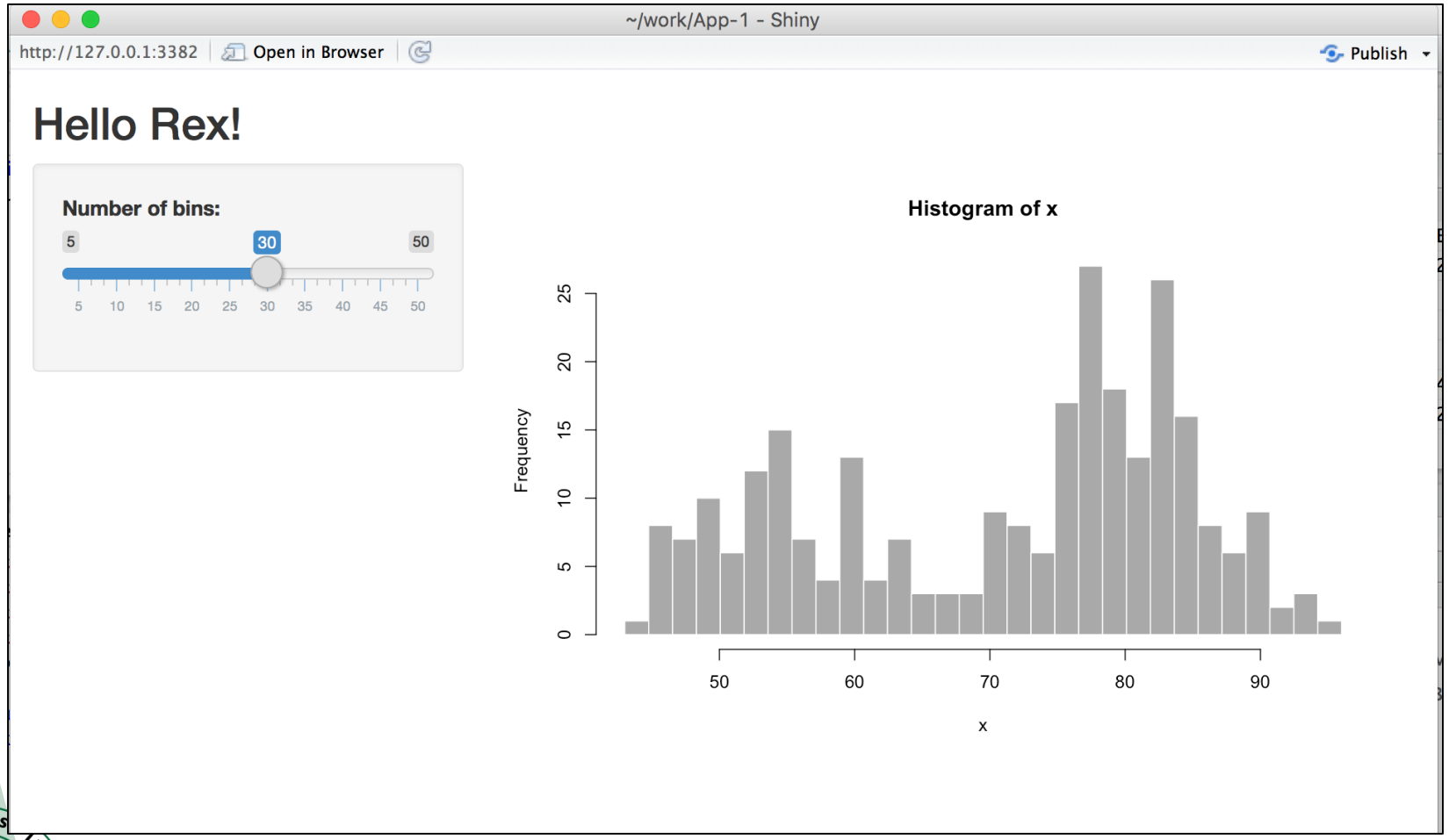
Shiny

- Create a **new R script**, type the following and Run.

```
library(shiny)
runApp("App-1")
```

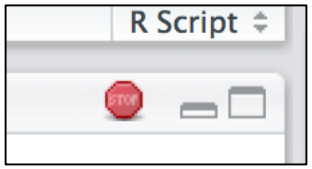


Shiny


~/work/App-1 - Shiny
http://127.0.0.1:3382
Open in Browser
Publish
Hello Rex!
Number of bins:
5
30
50
5 10 15 20 25 30 35 40 45 50
Histogram of x
Frequency
0 5 10 15 20 25
50 60 70 80 90
x

Shiny

- Closing the App
  - If done debugging, the App needs to be closed such that the R console will be free
  - Click on the Stop Button to stop the execution of the App



Building a Shiny App

- To get started, open the previous server.R and ui.R files.
- Edit the scripts to match the ones in the next two slides

Building a Shiny App

Example 6.3: Building from Scratch

- Open the server.R file, edit it the based on the following contents.

```
library(shiny)
shinyServer(function(input, output) {
})
```

```
1  library(shiny)
2  shinyServer(function(input, output) {
3  })
```

Building a Shiny App

- Open the ui.R file, edit it the based on the following contents.

```
library(shiny)
shinyUI(fluidPage(
))
```

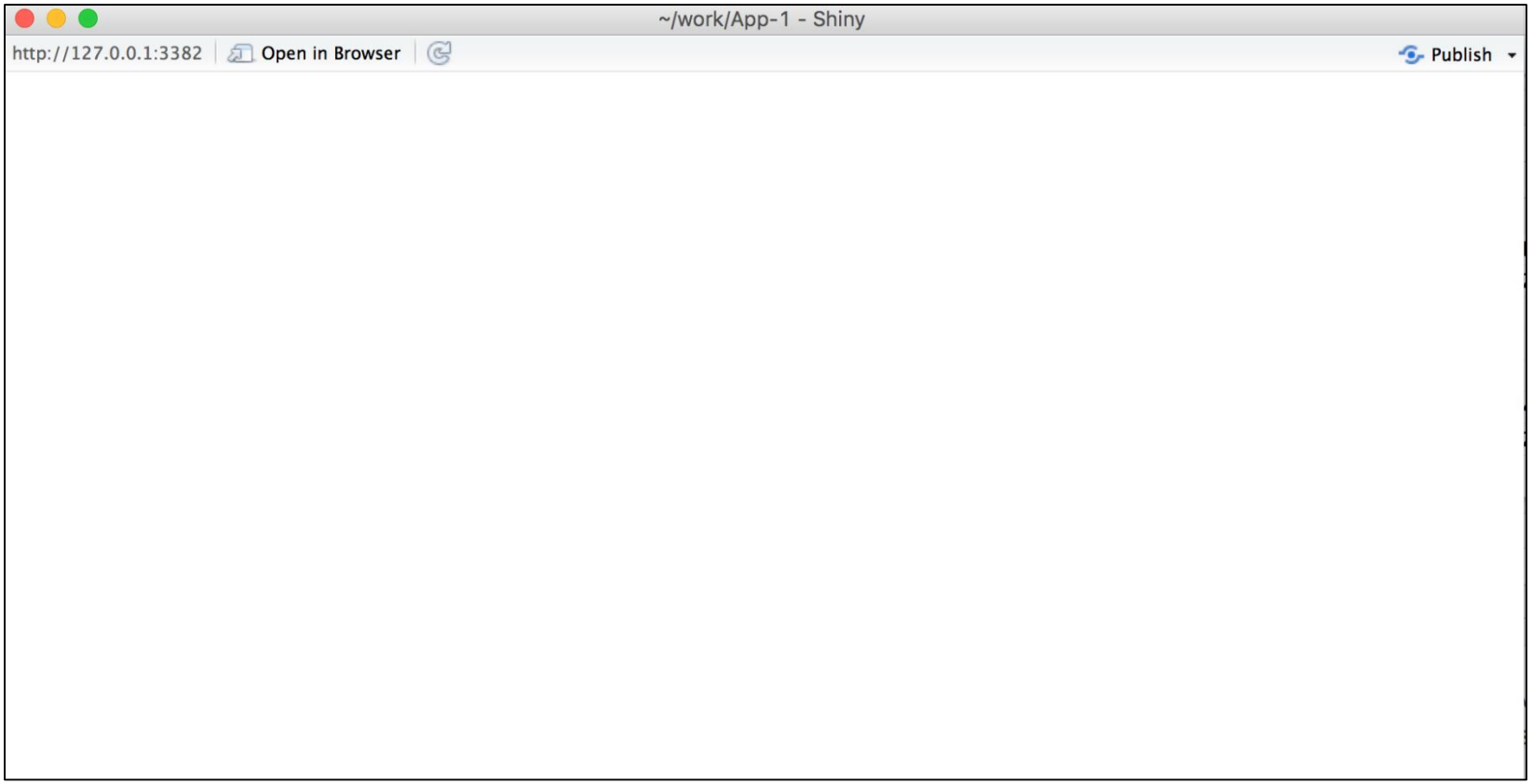
```
1  library(shiny)
2  shinyUI(fluidPage(
3  ))
```

Building a Shiny App

- Run the App

```
1  library(shiny)
2  runApp("App-1")
```

Building a Shiny App



Building a Shiny App

- Layout
  - Shiny ui.R scripts use the function **fluidPage** to create a display that automatically adjusts to the dimensions of your user's browser window.
  - A layout can be developed by **placing elements** in the fluidPage function.
  - For example, the ui.R in the next slide creates a user-interface that has a title panel and then a sidebar layout, which includes a sidebar panel and a main panel.
  - Note that these elements are placed **within the** fluidPage function.

Building a Shiny App

- Open the ui.R file, edit it the based on the following contents.

```
1  library(shiny)
2  shinyUI(fluidPage(
3    titlePanel("title panel"),
4    sidebarLayout(
5      sidebarPanel( "sidebar panel"),
6      mainPanel("main panel")
7    )
8  ))
```

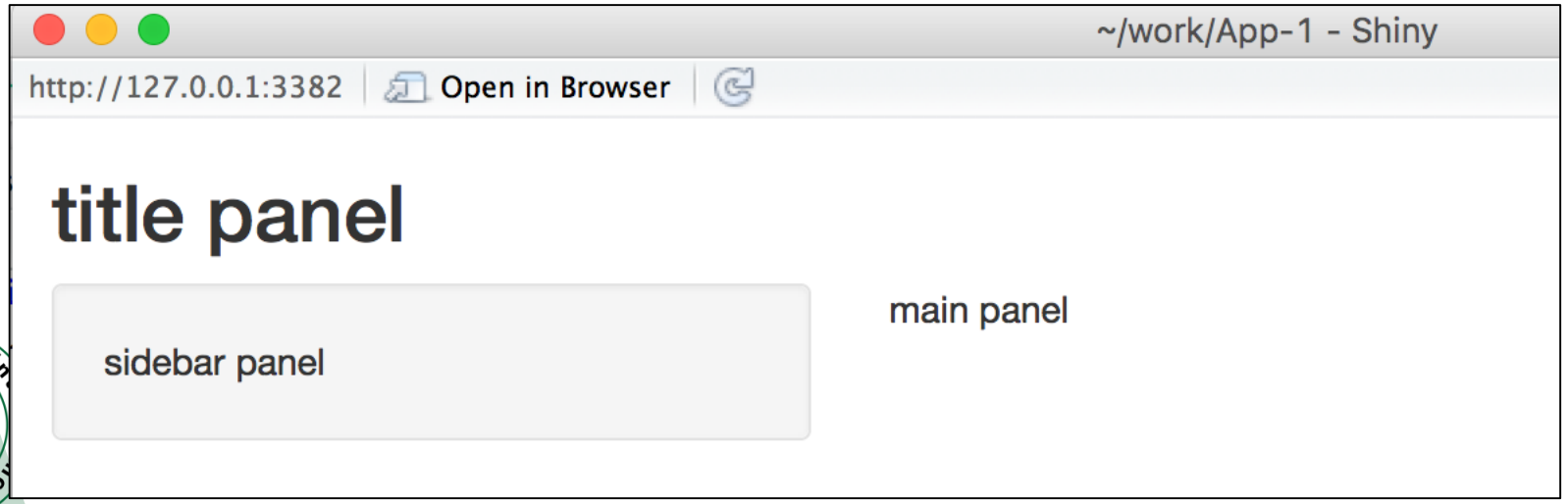
Building a Shiny App

- Run the App

```
1  library(shiny)
2  runApp("App-1")
```

Building a Shiny App

```
1  library(shiny)
2  shinyUI(fluidPage(
3    titlePanel("title panel"),
4    sidebarLayout(
5      sidebarPanel( "sidebar panel"),
6      mainPanel("main panel")
7    )
8  ))
```



Building a Shiny App

- The fluid Layout
  - titlePanel and sidebarLayout are the **two most popular elements** to add to fluidPage.
    - They create a basic Shiny app with a sidebar.
  - sidebarLayout always takes two arguments:
    - sidebarPanel function output
    - mainPanel function output
  - These functions **place content** in either the sidebar or the main panels. The sidebar panel will appear on the left side of your app by default. You can move it to the right side by giving sidebarLayout the optional argument position = "right".

Building a Shiny App

- You can add HTML content to your Shiny app by placing it inside a Panel function.

Shiny Function	HTML5 equivalent	Creates
p	<p>	A paragraph of text
h1	<h1>	A first level header
h2	<h2>	A second level header
h3	<h3>	A third level header
h4	<h4>	A fourth level header
h5	<h5>	A fifth level header
h6	<h6>	A sixth level header
a	<a>	A hyper link
br	 	A line break (e.g. a blank line)

Building a Shiny App

Shiny Function	HTML5 equivalent	Creates
div		A division of text with a uniform style
span		An in-line division of text with a uniform style
pre	<pre>	Text 'as is' in a fixed width font
code	<code>	A formatted block of code
img	<img>	An image
strong		Bold text
em		Italicized text
HTML		Directly passes a character string as HTML code

Building a Shiny App

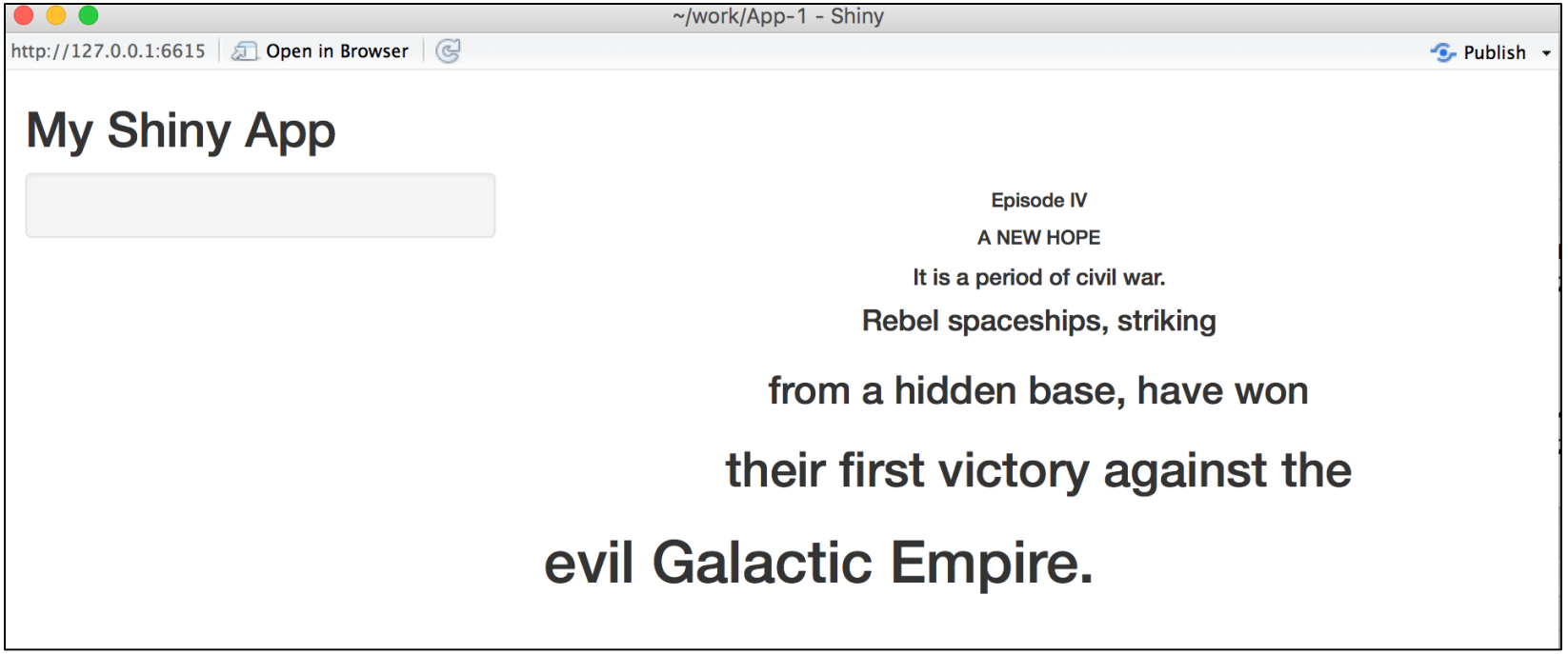
Example 6.4: HTML Content

- Open the ui.R file, edit it the based on the next slide contents

Building a Shiny App

```
1  library(shiny)
2  shinyUI(
3    fluidPage(
4      titlePanel("My Shiny App"),
5      sidebarLayout(
6        sidebarPanel(),
7        mainPanel(
8          h6("Episode IV", align = "center"),
9          h6("A NEW HOPE", align = "center"),
10         h5("It is a period of civil war.", align = "center"),
11         h4("Rebel spaceships, striking", align = "center"),
12         h3("from a hidden base, have won", align = "center"),
13         h2("their first victory against the", align = "center"),
14         h1("evil Galactic Empire.")
15       )
16     )
17   )
18 )
```

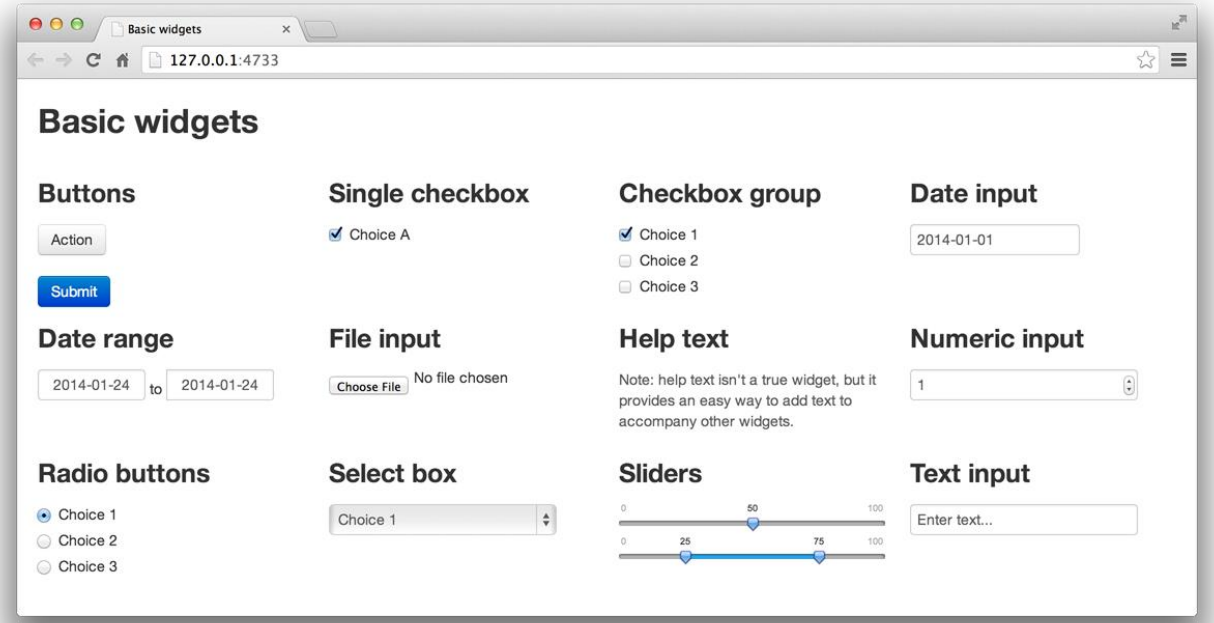
Building a Shiny App



Building a Shiny App

Definition 6.2: Control Widgets

- Widgets provide a way for your users to **interact with** the Shiny app.
- Shiny widgets collect a value from your user. When a user changes the widget, the value will change as well.



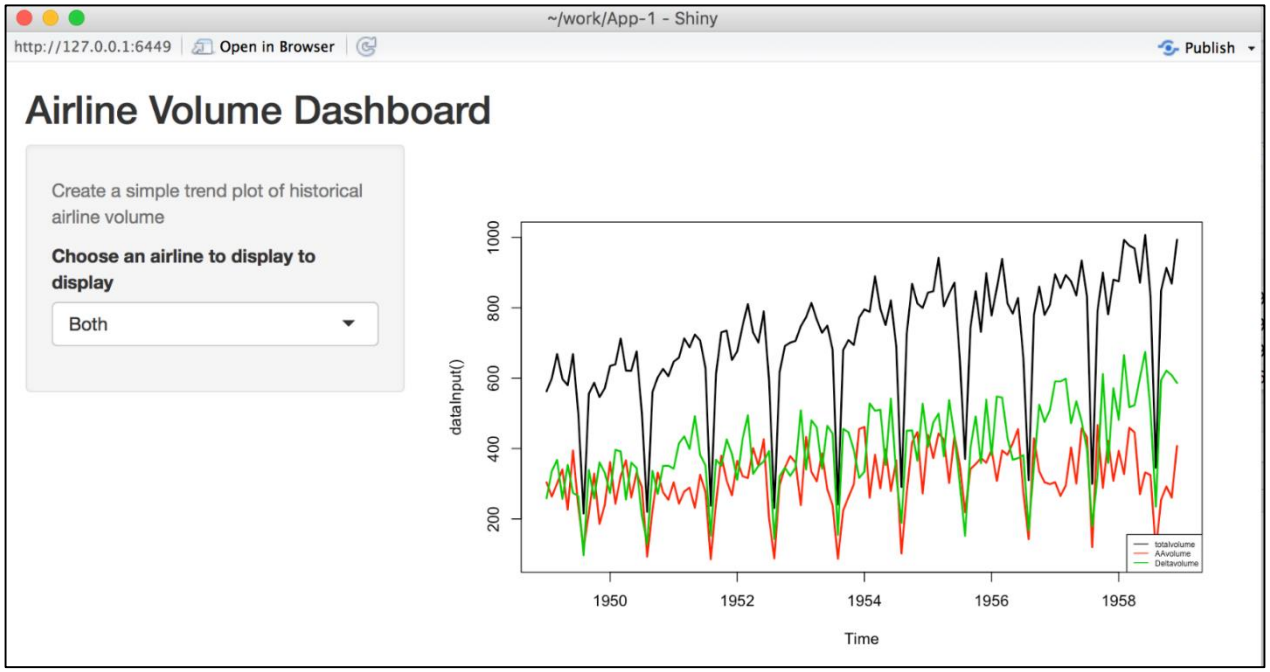
Building a Shiny App

function	widget
actionButton	Action Button
checkboxGroupInput	A group of check boxes
checkboxInput	A single check box
dateInput	A calendar to aid date selection
dateRangeInput	A pair of calendars for selecting a date range
fileInput	A file upload control wizard
helpText	Help text that can be added to an input form
numericInput	A field to enter numbers
radioButtons	A set of radio buttons
selectInput	A box with choices to select from
sliderInput	A slider bar
submitButton	A submit button
textInput	A field to enter text

Building a Shiny App

Example 6.5: Building a Shiny App

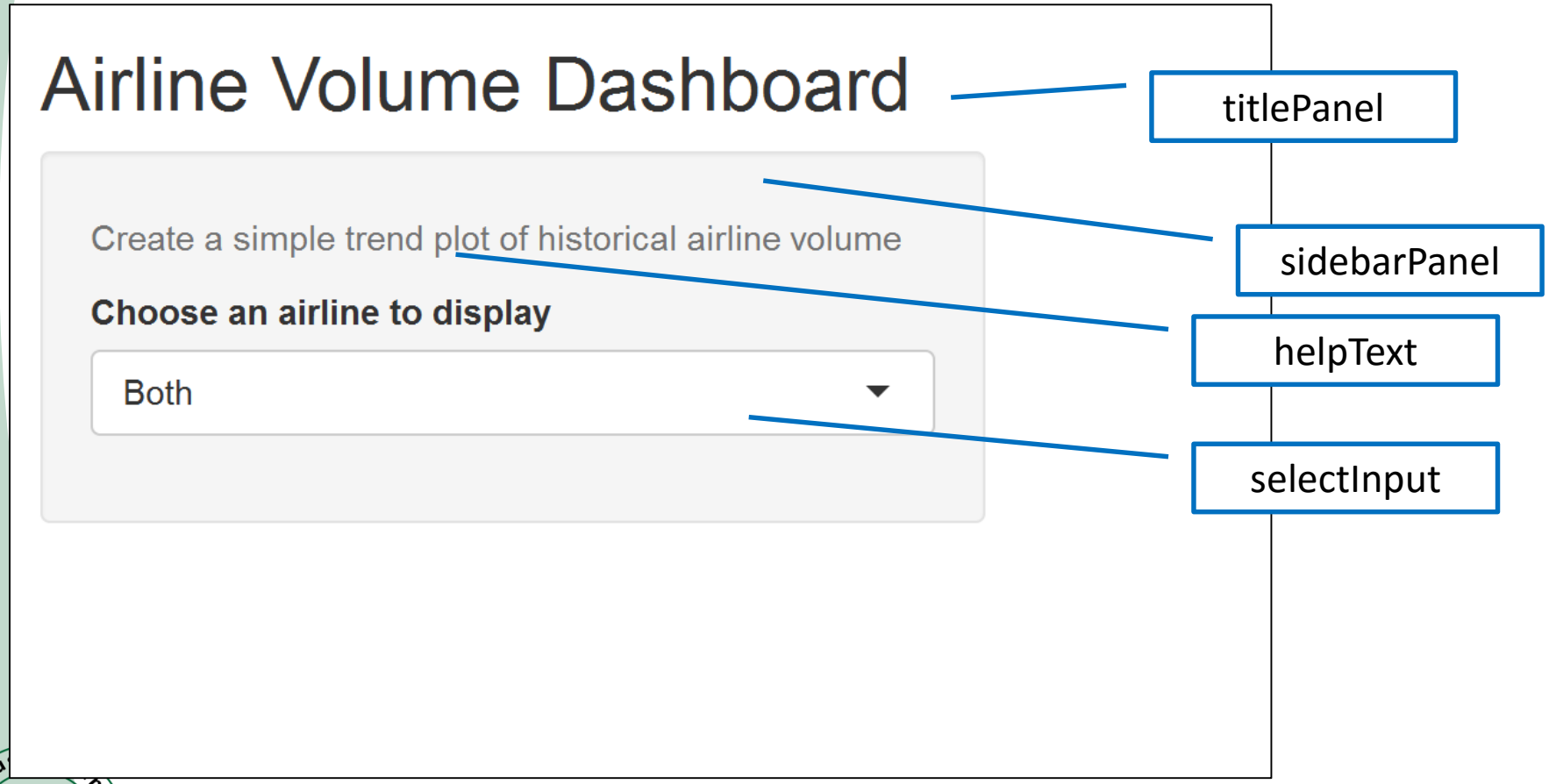
- Rewrite the ui.R script to create the user-interface displayed below. This Shiny app uses a basic Shiny layout (no columns) and contains widgets. The other values of the select box are shown below the image of the app.



Building a Shiny App

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Airline Volume Dashboard"),
  sidebarLayout(
    sidebarPanel(
      helpText("Create a simple trend plot of historical airline volume"),
      selectInput("var",  label = "Choose an airline to display",
                  choices = list("Both", "American Airlines","Delta"),
                  selected = "Both")
    ),
    mainPanel(
    )
  )
))
```

Building a Shiny App



Reactive Output

Definition 6.3: Reactive Output

- Reactive output **automatically responds** when a user toggles a widget.
- Two Steps:
  - Add an R object to your user-interface with ui.R.
  - Tell Shiny how to build the object in server.R. The object will be reactive if the code that builds it calls a widget value.

Reactive Output

- Step 1: Add an R object to the UI
  - Shiny provides a family of functions that turn R objects into output for your user-interface.
  - Each function creates a specific type of output.

Output Function	**Creates**
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

Reactive Output

Example 6.5: Reactive Output Example

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Airline Volume Dashboard"),
  sidebarLayout(
    sidebarPanel(
      helpText("Create a simple trend plot of historical airline volume"),
      selectInput("var",  label = "Choose an airline to display",
                  choices = list("Both", "American Airlines","Delta"),
                  selected = "Both")
    ),
    mainPanel(
      textOuput("text1")
    )
  )
))
```

Reactive Output

- Step 2: Provide R code to build the object.
  - Need to tell Shiny **how to build** the text object.
  - Do this by providing R code that builds the object in server.R.
  - The code should go in the unnamed function that appears inside shinyServer in your server.R script.
  - The unnamed function plays a special role in the Shiny process; it builds a list-like object named output that contains all of the code needed to update the R objects in your app.
  - Each R object needs to have its own entry in the list.

Reactive Output

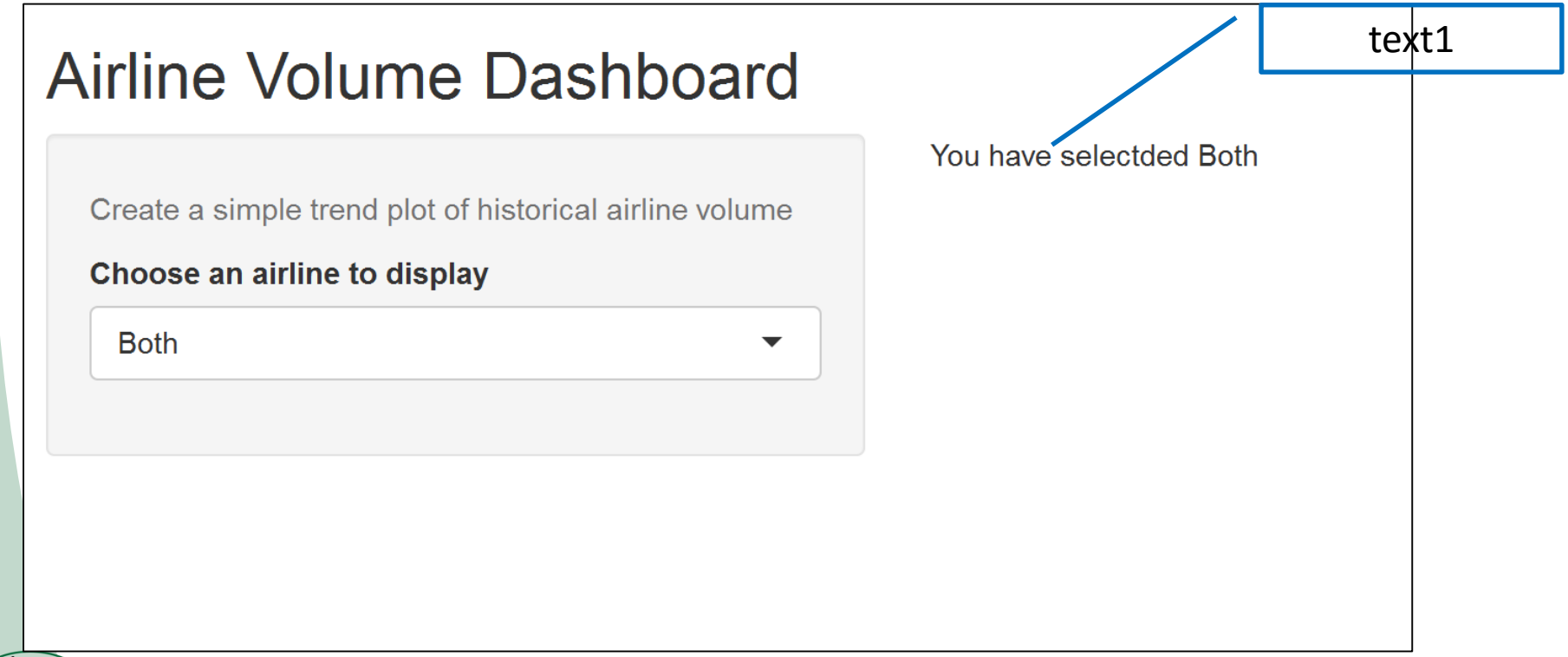
Render Function	Creates
renderImage	images (saved as a link to a source file)
renderPlot	plots
renderPrint	any printed output
renderTable	data frame, matrix, other table like structures
renderText	character strings
renderUI	a Shiny tag object or HTML

Reactive Output

Example 6.5 (Cont.) : Reactive Output Example

```
1  library(shiny)
2  shinyServer(
3    function(input, output) {
4      output$text1 <- renderText({
5        paste("You have selected", input$var)
6      })
7    }
8  )
```

Reactive Output



Shiny Execution Options

- Shiny Execution: server.R

```
# server.R

# A place to put code

shinyServer(
  function(input, output) {

    # Another place to put code

    output$map <- renderPlot({

        # A third place to put code

    })

  }
)
```

Run once when app is launched

Shiny Execution Options

- Shiny Execution: server.R

```
# server.R

# A place to put code

shinyServer(
  function(input, output) {

    # Another place to put code

    output$map <- renderPlot({

        # A third place to put code

    })

  }
)
```

Run once each time a user visits the app

Shiny Execution Options

- Shiny Execution: server.R

```
# server.R

# A place to put code

shinyServer(
  function(input, output) {

    # Another place to put code

    output$map <- renderPlot({

      # A third place to put code

    })

  }
)
```

Run each time a user changes a widget that output$map relies on

Shiny Execution Options

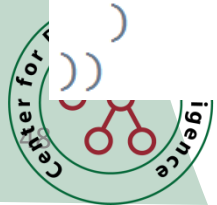
Example 6.6: Execution Options

- Copy the file Airline.csv from the “for sharing” folder and paste it into the App-1 folder
- Modify the ui.R and server.R codes as follows in the next slides

Shiny Execution Options

Example 6.6 (Cont.): Execution Options

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Airline Volume Dashboard"),
  sidebarLayout(
    sidebarPanel(
      helpText("Create a simple trend plot of historical airline volume"),
      selectInput("var",  label = "Choose an airline to display",
                  choices = list("Both", "American Airlines","Delta"),
                  selected = "Both")
    ),
    mainPanel(
      plotOutput("plot1")
    )
  )
))
```



Shiny Execution Options

Example 6.6 (Cont.): Execution Options

```
1  library(shiny)
2  library("TTR")
3  tempdata = read.csv("airline.csv")
4  totalvolume = ts(tempdata[,4], frequency=12, start=c(1949,1))
5  AAvolume = ts(tempdata[,5], frequency=12, start=c(1949,1))
6  Deltavolume = ts(tempdata[,6], frequency=12, start=c(1949,1))
7  total = cbind(totalvolume,AAvolume,Deltavolume)
```

Shiny Execution Options

Example 6.6 (Cont.): Execution Options

```
8  shinyServer(
9    function(input, output) {
10     dataInput <- reactive({
11       switch(input$var,
12              "Both" = total,
13              "American Airlines" = AAvolume,
14              "Delta" = Deltavolume
15       )
16     })
17     colorInput <- reactive({
18       switch(input$var,
19              "Both" = 1:3,
20              "American Airlines" = 2,
21              "Delta" = 3
22       )
23     })
```

Shiny Execution Options

Example 6.6 (Cont.): Execution Options

```
24      output$plot1 <- renderPlot({
25        plot(dataInput(), plot.type="single", col=colorInput(),
26             lwd = c(2, 2, 2,2))
27        legend("bottomright", colnames(total), col=1:ncol(total),
28             lty = c(1, 1, 1,1), cex=.5, y.intersp = 1)
29      })
30    }
31  )
```

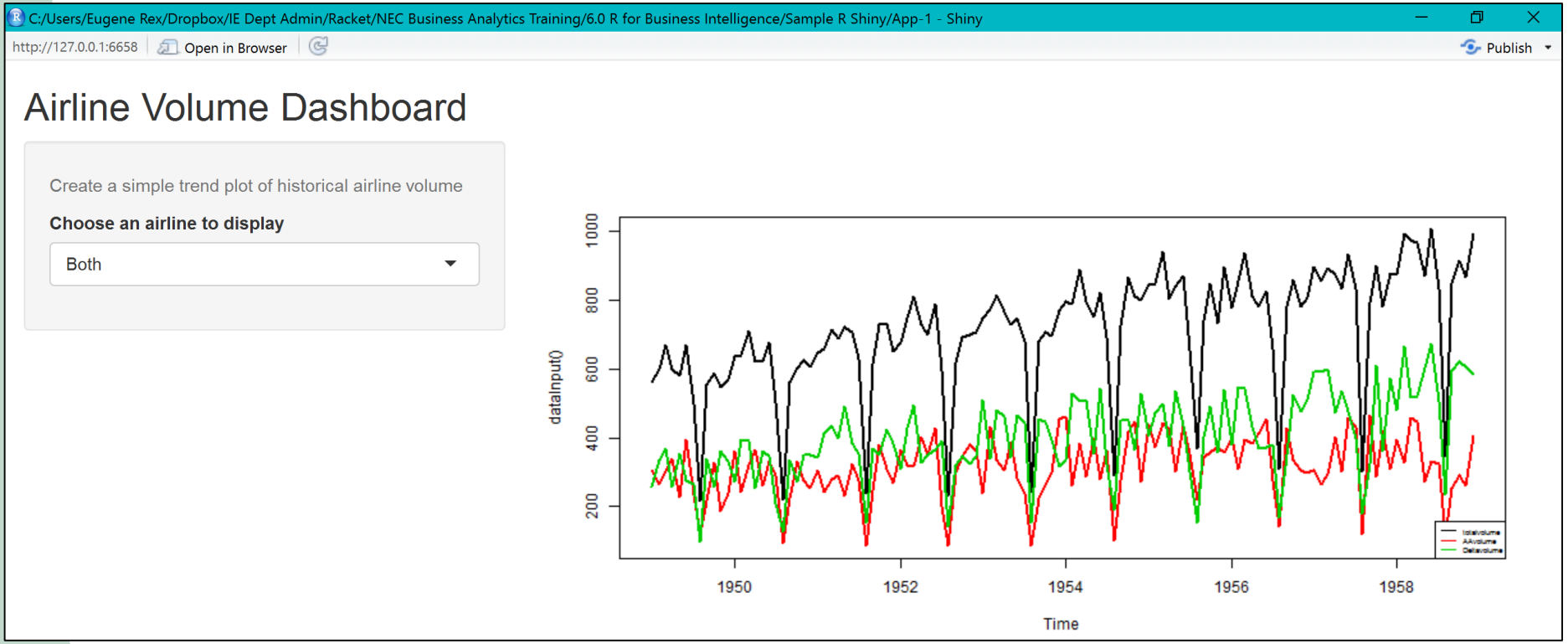
Shiny Execution Options

Example 6.6 (Cont.): Execution Options

- Run the App

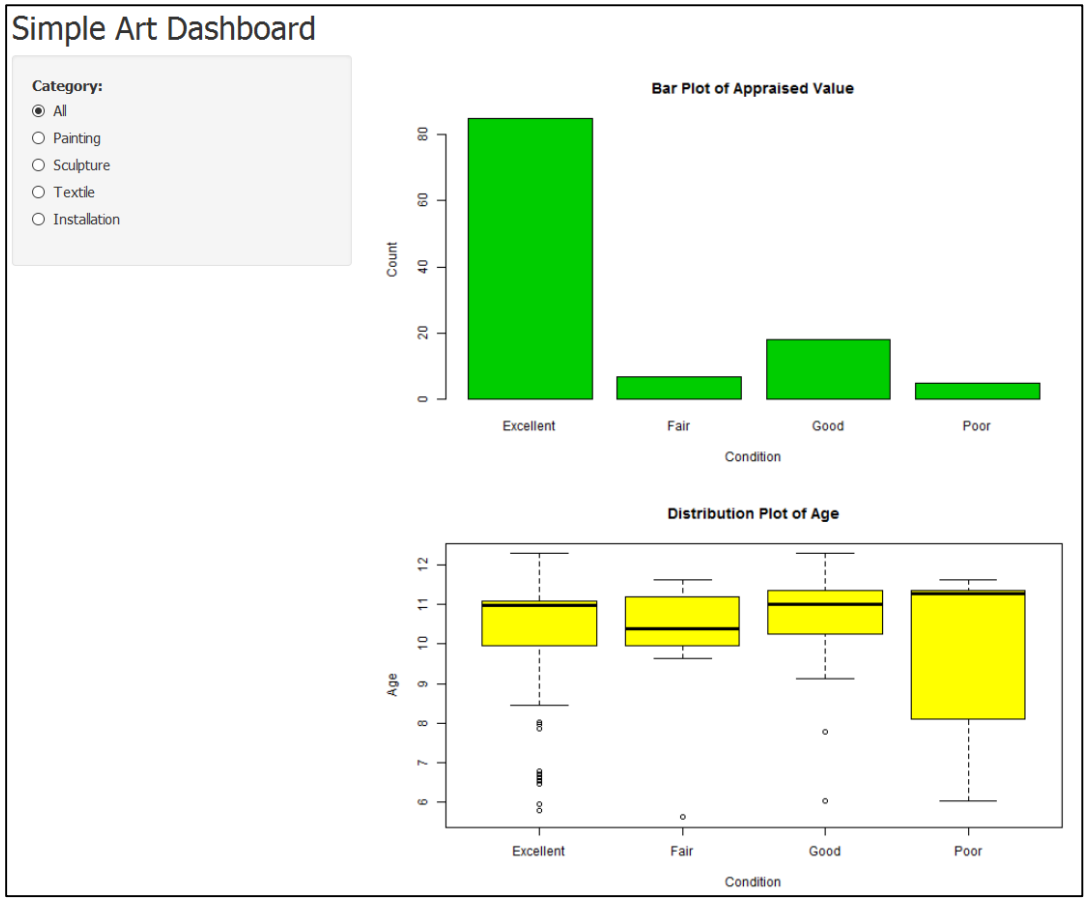
```
1  library(shiny)
2  runApp("App-1")
```

Shiny Execution Options



Case 4

- Simple dashboard generation from scratch



References

- http://shiny.rstudio.com/tutorial/