Introduction to Interactive Data Visualisation

using R Shiny

Royal Mail Data Science Workshop 17th October 2018

Ana Guedes

Royal Mail

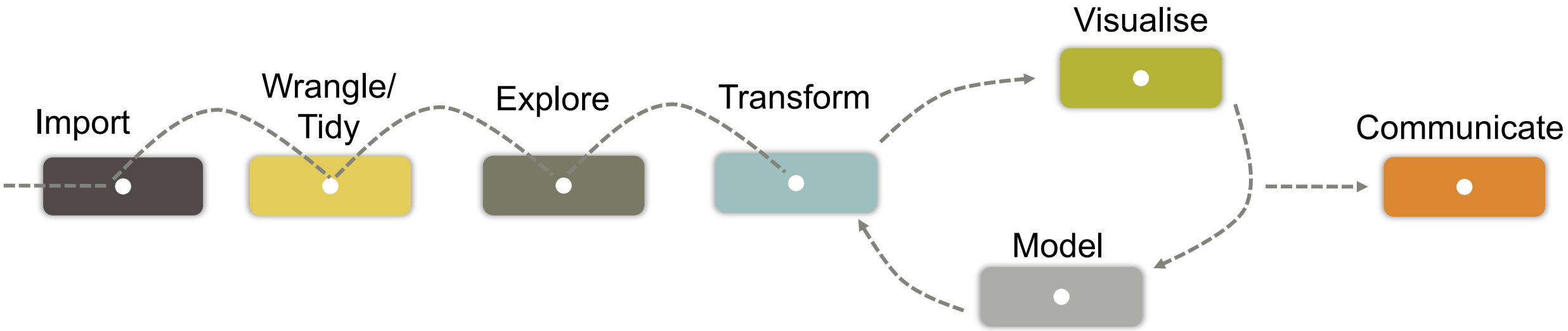
#RMdatasci

Goals for this workshop

1. Importance of **data visualisation**
2. Understand the **structure** of a *Shiny* app
3. **Create** a Shiny app

Introduction and General concepts

The Data Science process



Data visualisation is crucial in various steps of the data science process

http://r4ds.had.co.nz/introduction.html

Why interactive data visualisation?

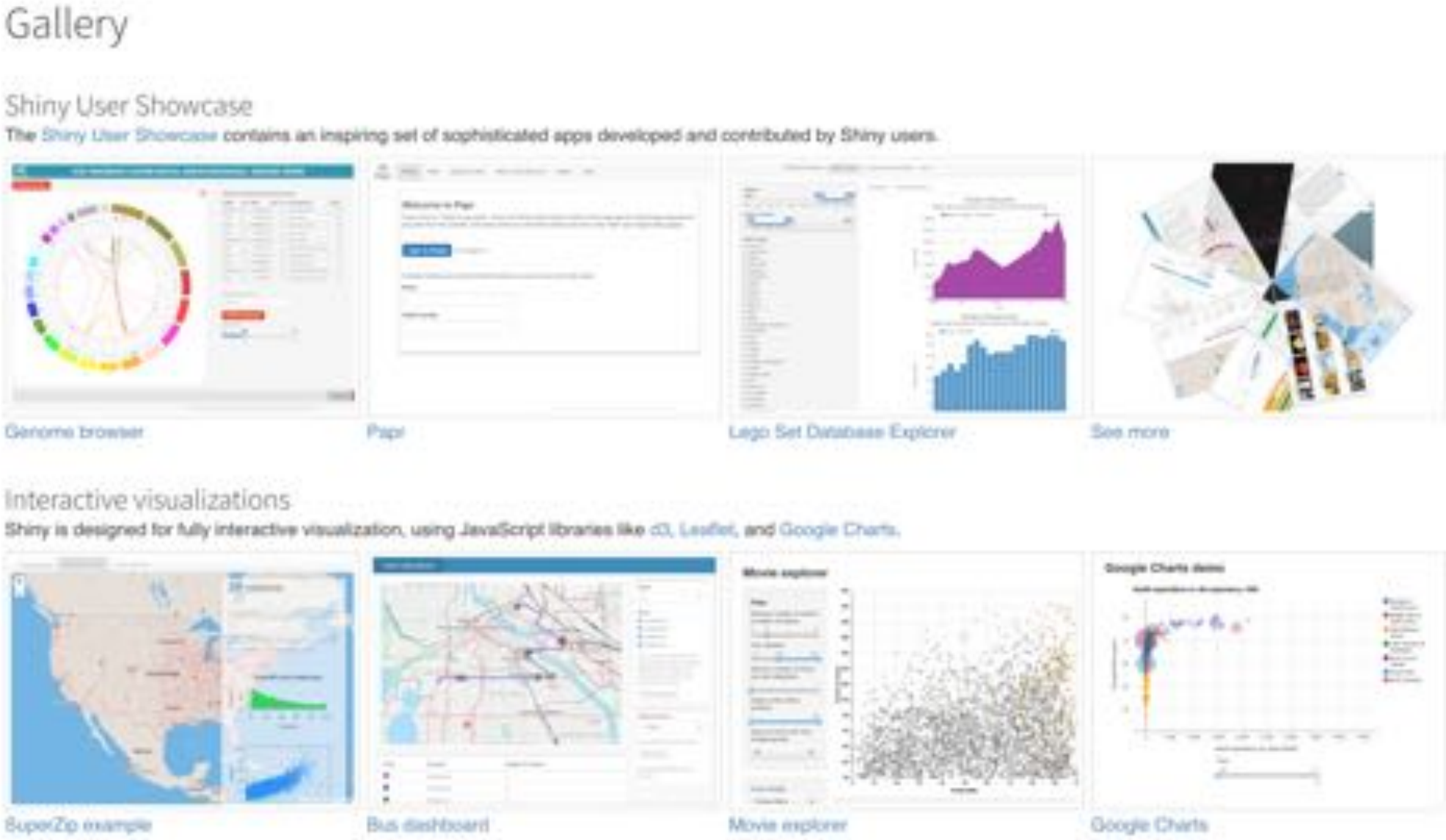
- More **engaging** and better user **experience**
- **Empowers** the user
- Telling a **story** through data
- Easier to understand and **remember**
- **Succinct** communication of data

Why R Shiny?

Web application framework for R that helps turn data analyses into interactive web applications

- **Open** source and **big community**
- Develop for the web using the **same language** as for data exploration and modelling
- No HTML, CSS, or JavaScript knowledge required!
- Pre-built widgets, allowing to build **elegant** and **powerful** applications with minimal effort

Shiny examples



http://shiny.rstudio.com/gallery/

Webapps

- **Web app:** application program stored on a remote server and accessed via a web browser.

Front-end

- Producing HTML, CSS and JavaScript for a website or web application so that a user can see and interact with them directly

Back-end

- Creates the logical back-end and core computational logic of a website

https://bookdown.org/weicheng/shinyTutorial/web-development.html

Building a shiny app

A Shiny app is composed of 2 parts:

- ✓ **UI:** web document that the user gets to see

User Interface (UI)

- ✓ **Server:** set of instructions that tell the web page what to show when the user interacts with the page

Server Instructions

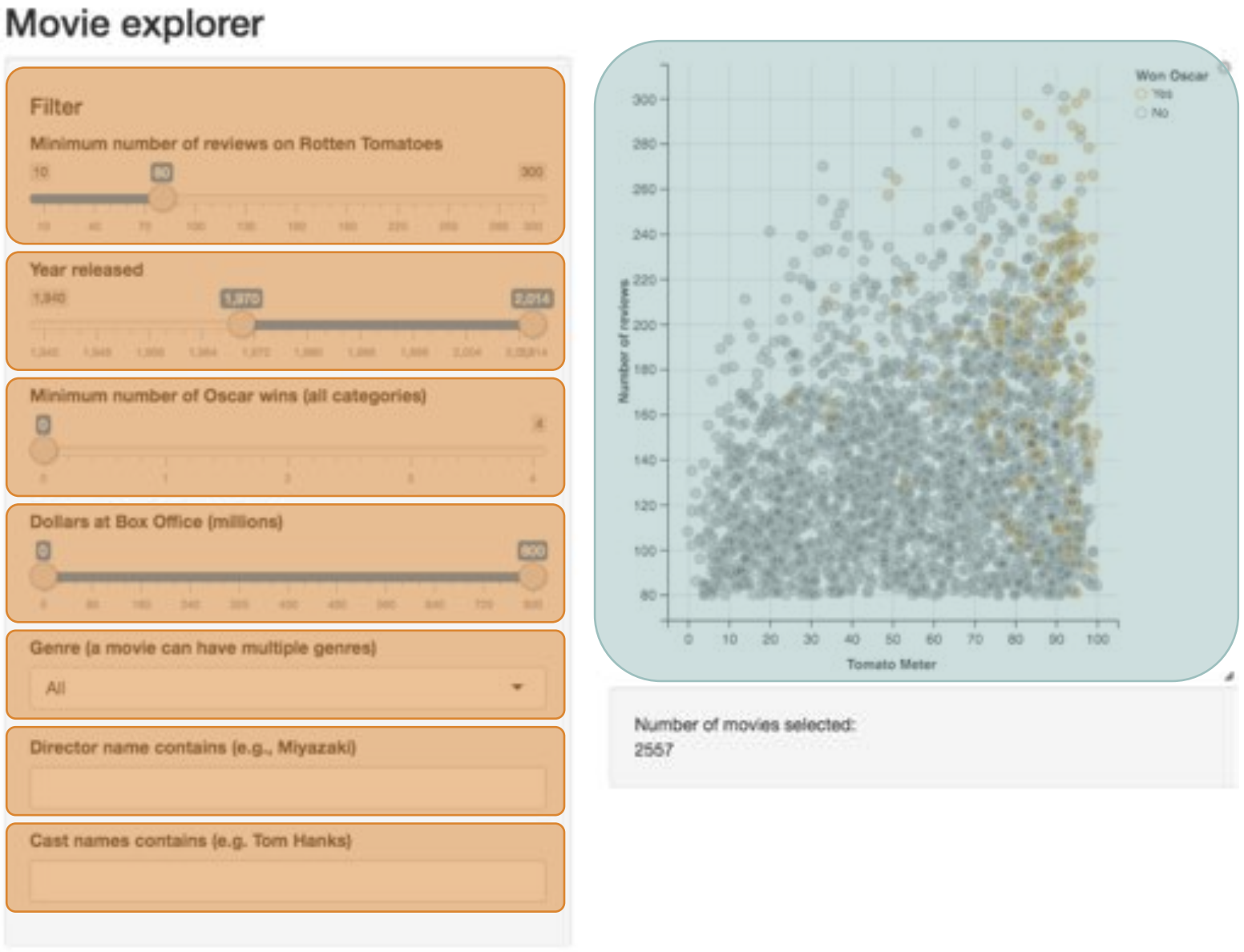
How does a Shiny app work?

- The *Server* keeps monitoring the *UI*. Whenever there is a change in the *UI*, the *Server* will follow some instructions (run some R code) accordingly and update the *UI* (concept of **reactivity**)

https://deanattali.com/blog/building-shiny-apps-tutorial/

Shiny components

- **Shiny apps are built around inputs and outputs**

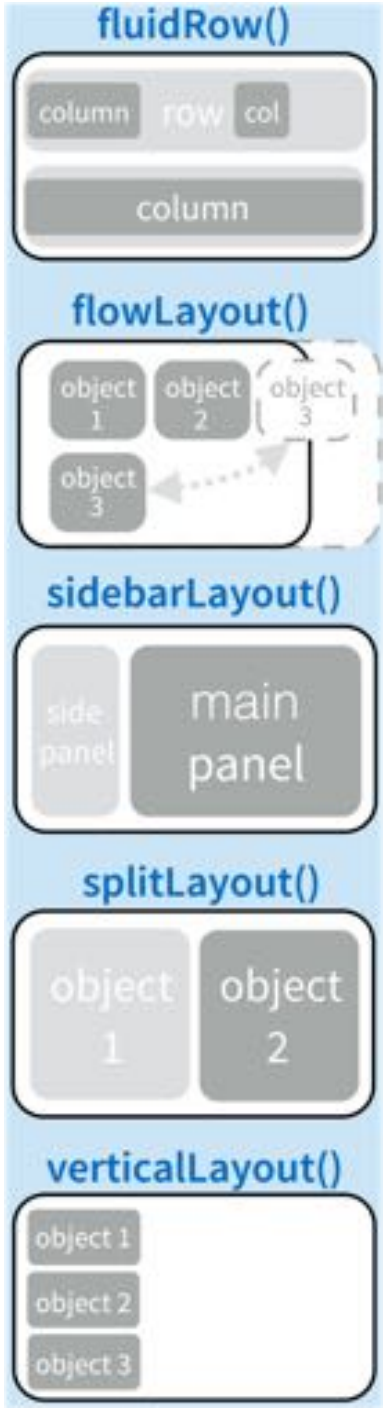


https://shiny.rstudio.com/gallery/movie-explorer.html

ui.R

1. Layout

- **Panels:** group elements together into a single ‘panel’.
- **Layouts:** organize panels and elements



UI Layout

Functions for laying out the user interface for your application.

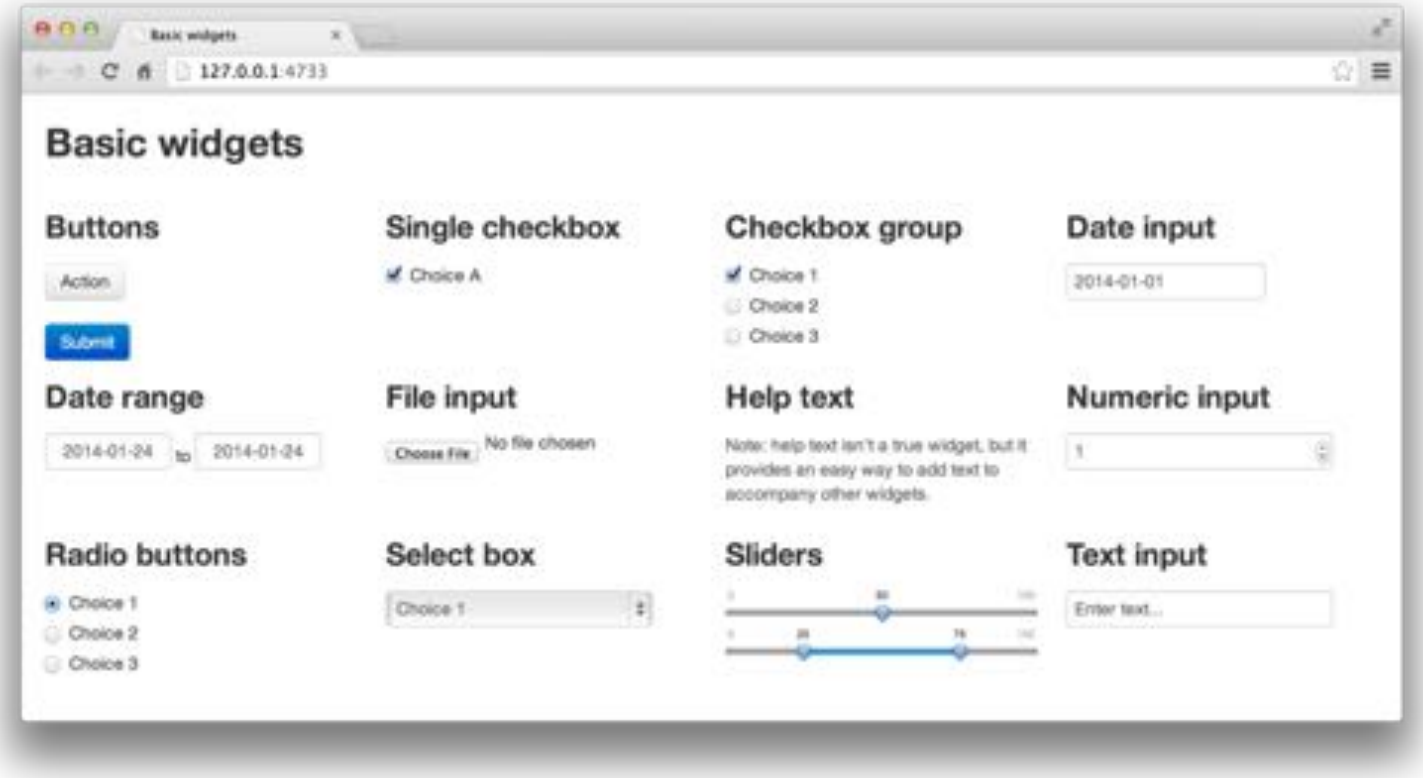
absolutePanel (fixedPanel)	Panel with absolute positioning
bootstrapPage (basicPage)	Create a Bootstrap page
column	Create a column within a UI definition
conditionalPanel	Conditional Panel
fillPage	Create a page that fills the window
fillRow (fillCol)	Flex Box-based row/column layouts
fixedPage (fixedRow)	Create a page with a fixed layout
fluidPage (fluidRow)	Create a page with fluid layout
headerPanel	Create a header panel
helpText	Create a help text element
icon	Create an icon
mainPanel	Create a main panel
navbarPage (navbarMenu)	Create a page with a top level navigation bar
navlistPanel	Create a navigation list panel
pageWithSidebar	Create a page with a sidebar
sidebarLayout	Layout a sidebar and main area
sidebarPanel	Create a sidebar panel
tabPanel	Create a tab panel
tabsetPanel	Create a tabset panel
titlePanel	Create a panel containing an application title.
inputPanel	Input panel
flowLayout	Flow layout
splitLayout	Split layout
verticalLayout	Lay out UI elements vertically

https://shiny.rstudio.com/reference/shiny/1.0.5/
https://bookdown.org/weicheng/shinyTutorial/ui.html

ui.R

2. Input

- Create with an ***Input()** functions
- Adding widgets/inputs
  - **Name (id)**
  - **Label**
  - Called using ***input$name***




```
sliderInput(inputId = "myid", label = "this will show in the UI", …)
```

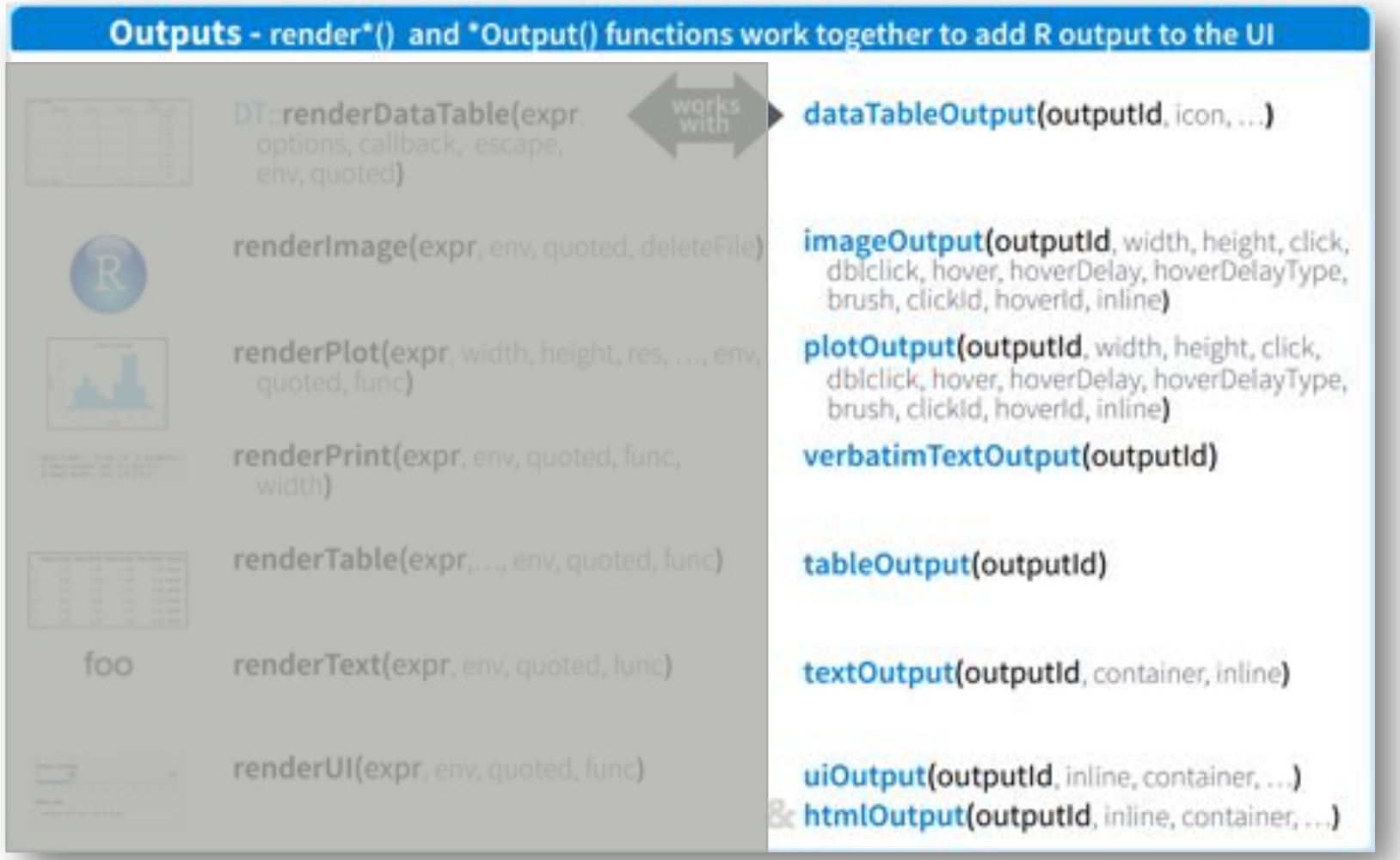
https://shiny.rstudio.com/reference/shiny/1.0.5/
https://bookdown.org/weicheng/shinyTutorial/ui.html

ui.R

3. Output

- Create with ***Output()** functions
- ***Output()** adds a space in the ui.R for an R object

```
plotOutput("histogram")
```

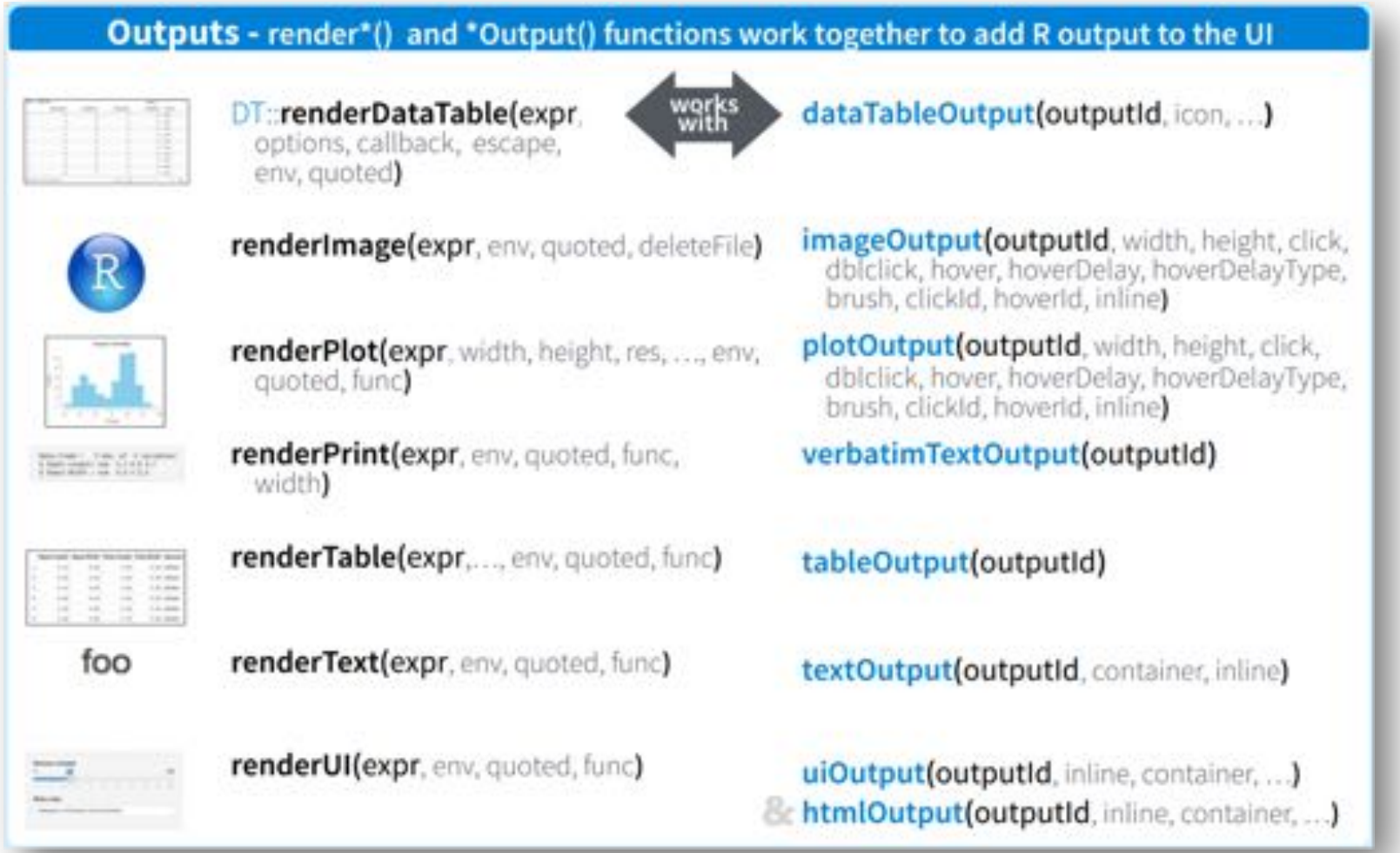


https://shiny.rstudio.com/reference/shiny/1.0.5/
https://bookdown.org/weicheng/shinyTutorial/ui.html

server.R

- Tells the server **how** to render logic using **render*()**
- Works together with ***output()**

```
Output$histogram =
renderPlot({
    hist(rnorm(100))
})
```

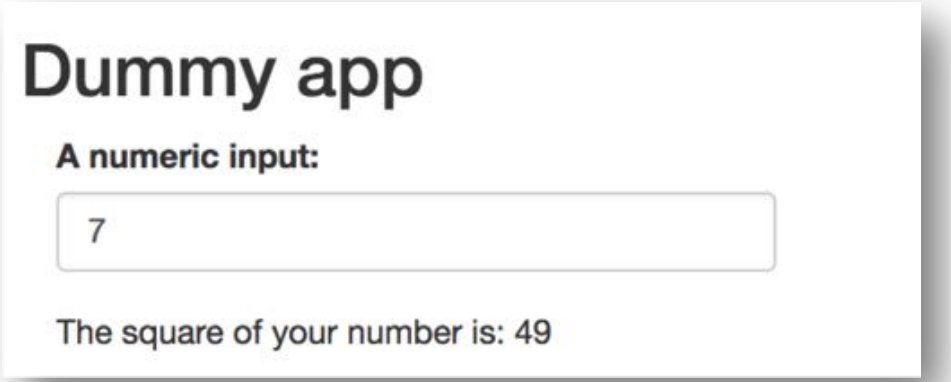


https://shiny.rstudio.com/reference/shiny/1.0.5/
https://bookdown.org/weicheng/shinyTutorial/ui.html

Dummy example

```
ShinyUI(fluidPage(
        titlePanel("Dummy app"),
        mainPanel(
           numericInput(id = "numInput", label = "A numeric input:",
                        value = 7, min = 1, max = 30),
           textOutput("squareNum")
        )
))
```

```
ShinyServer(function(input, output) {
 output$squareNum = renderText({
    paste("The square of your number is: ",
    input$numInput^2)
    })
})
```



Watch out for „ **( ) { }**
When your component is as an argument to a function, you should use „ otherwise don't!
Use „ inside a fluidRow. Don't use „ to separate elements in server.R file.

Reactivity 101

Reactivity is what enables your outputs to react to changes in inputs. If x is a reactive variable, this means that when the value of a variable x changes, then anything that relies on x gets re-evaluated

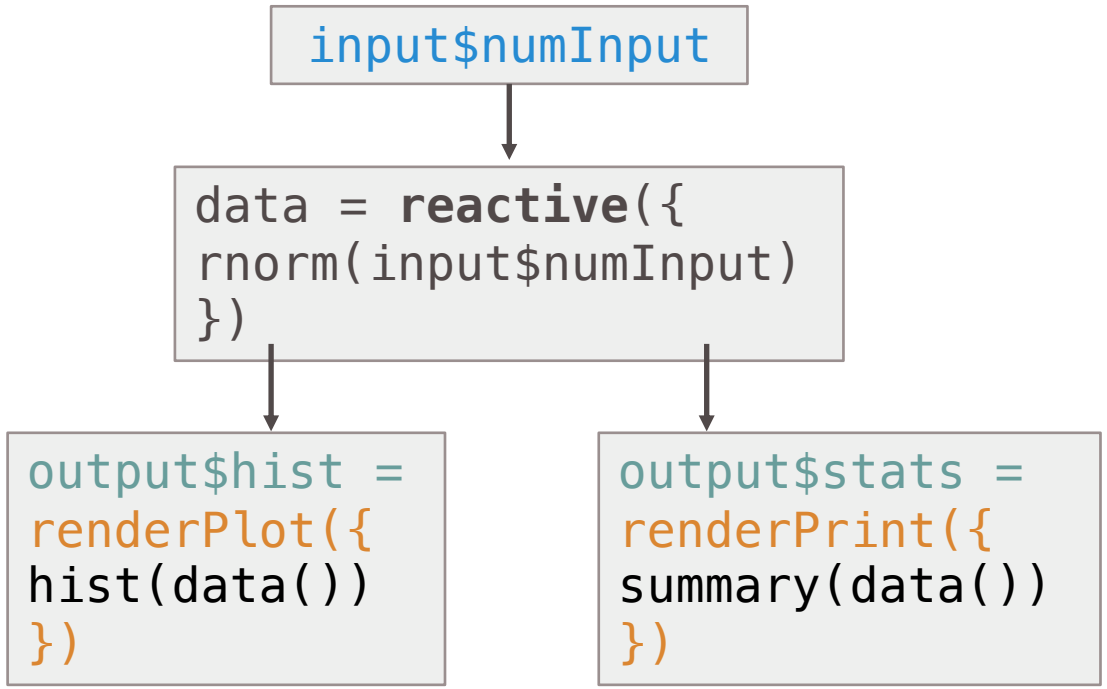
- In Shiny all **inputs** are automatically **reactive**.
- Reactive variables can only be used inside reactive contexts. Any **render*** function is a reactive context
- Besides render*(), there are other 2 common reactive contexts: **reactive({ })** and **observe({ })**

Whenever you want to print a reactive variable for debug you need to remember to wrap it in an observe({ }):
observe({ print(input$x)})

http://rstudio.github.io/shiny/tutorial/#reactivity
https://deanattali.com/blog/building-shiny-apps-tutorial/#10-reactivity-101

Reactivity

- Modularize code with **reactive()**
- **Reactive expression**
  1. Call a reactive expression like a function. ex. data()
  2. Reactive expression cache their values to avoid unnecessary computation



http://rstudio.github.io/shiny/tutorial/#reactivity
https://deanattali.com/blog/building-shiny-apps-tutorial/#10-reactivity-101

Summary

User Interface (UI)

Server Instructions

- Create some inputs
- Create a space for output

- Choose a reactive function that will accept our input
- Assign the reactive function to an output that will update the user interface

Share your app

- **shinyapps.io** – a server maintained by Rstudio

shinyapps.io by RStudio

http://docs.rstudio.com/shinyapps.io/getting-started.html

- **Shiny Server** – build your own server

Shiny Server

https://shiny.rstudio.com/articles/shiny-server.html

- **Docker container** – deploy and share anywhere

(wait for one of our next workshops to know learn more about docker!)

rocker/shiny ☆

How to Build a Shiny App – Demo

Repo available here: **https://github.com/royalmail-datascience/Meetup1_DataVisualisationShiny.git**

Hands on

- Create a Shiny app to visualise the 500 best albums of all time by *Rolling Stone*

Number	Year	Album	Artist	Genre	Subgenre
1	1967	Sgt. Pepper's Lonely Hearts Club Band	The Beatles	Rock	Rock & Roll, Psychedelic Rock
2	1966	Pet Sounds	The Beach Boys	Rock	Pop Rock, Psychedelic Rock
3	1966	Revolver	The Beatles	Rock	Psychedelic Rock, Pop Rock
4	1965	Highway 61 Revisited	Bob Dylan	Rock	Folk Rock, Blues Rock
5	1965	Rubber Soul	The Beatles	Rock, Pop	Pop Rock
6	1971	What's Going On	Marvin Gaye	Funk / Soul	Soul
7	1972	Exile on Main St.	The Rolling Stones	Rock	Blues Rock, Rock & Roll, Classic Rock
8	1979	London Calling	The Clash	Rock	Punk, New Wave
9	1966	Blonde on Blonde	Bob Dylan	Rock, Blues	Folk Rock, Rhythm & Blues
10	1968	The Beatles ("The White Album")	The Beatles	Rock	Rock & Roll, Pop Rock, Psychedelic Rock, Experimental

What do we want to know?

- ✓ What's the best artist of all time? And in a specific date interval?
- ✓ Which years/artists were more prolific in best albums?
- ✓ How has genre/artist popularity evolved along time?

https://www.kaggle.com/notgibs/500-greatest-albums-of-all-time-rolling-stone

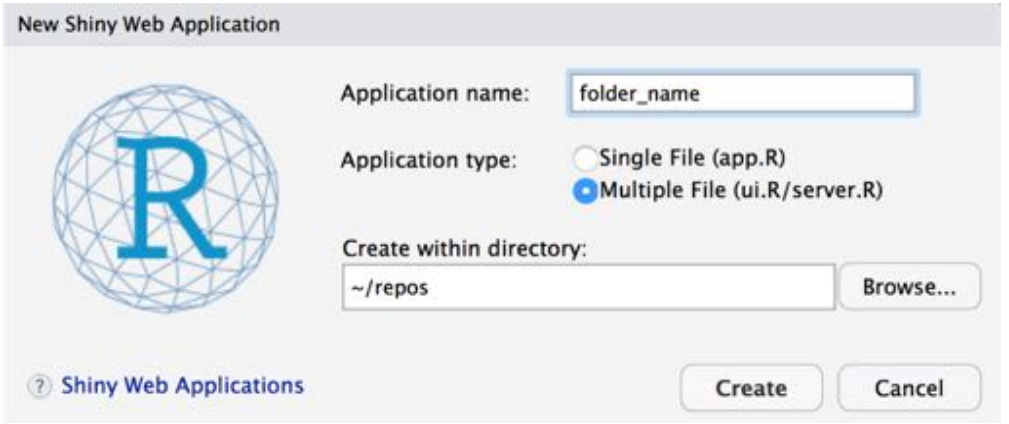
How to get started?

1. Open RStudio

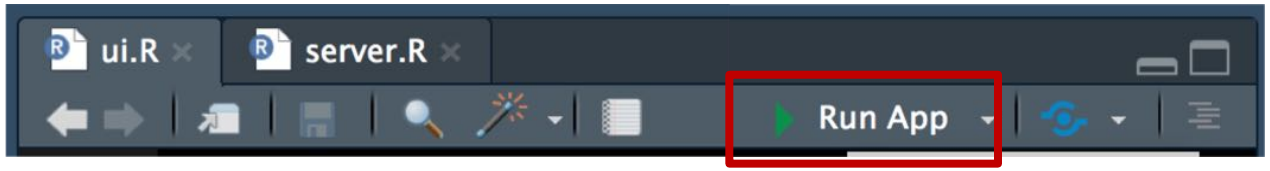
2.
```
install.packages("shiny")
library(shiny)
```

3. Create an empty Shiny app

3.1 File → New → Shiny Web App → Multiple File

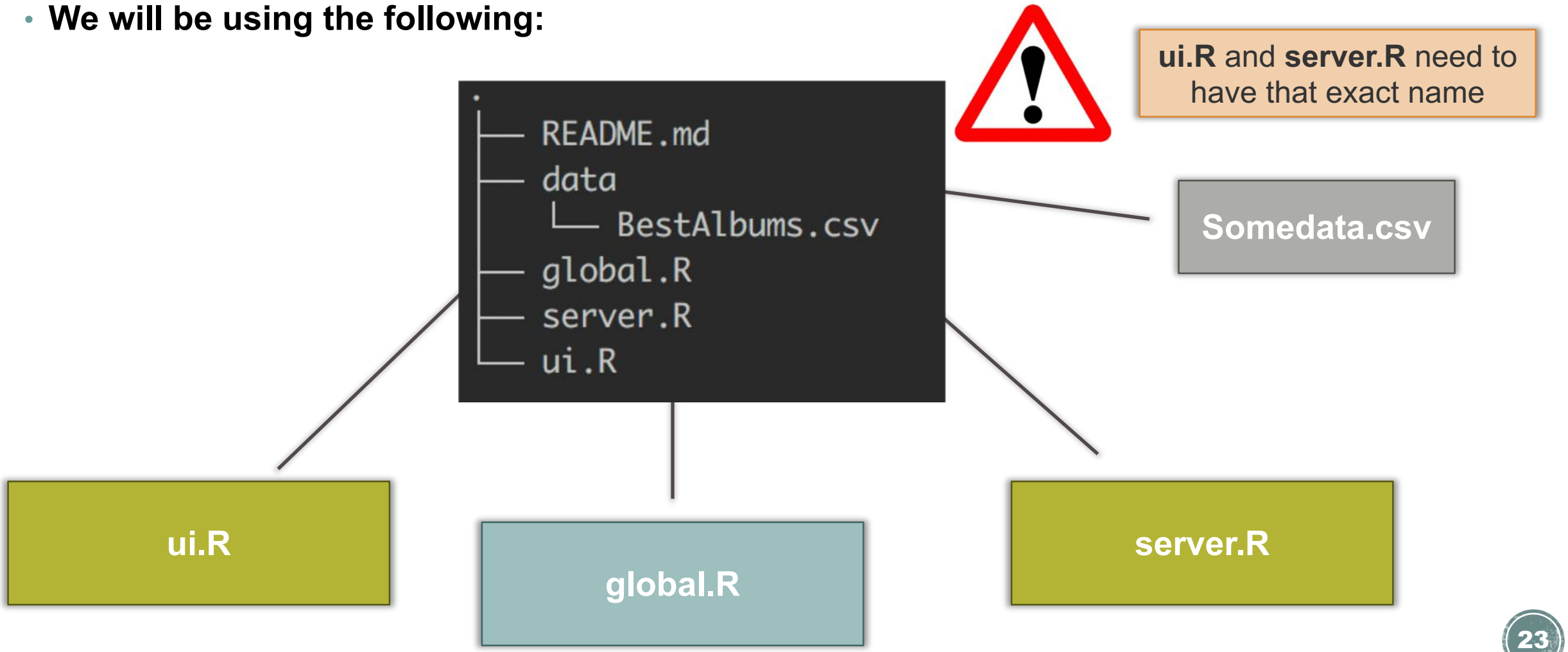


3.2 Run the app

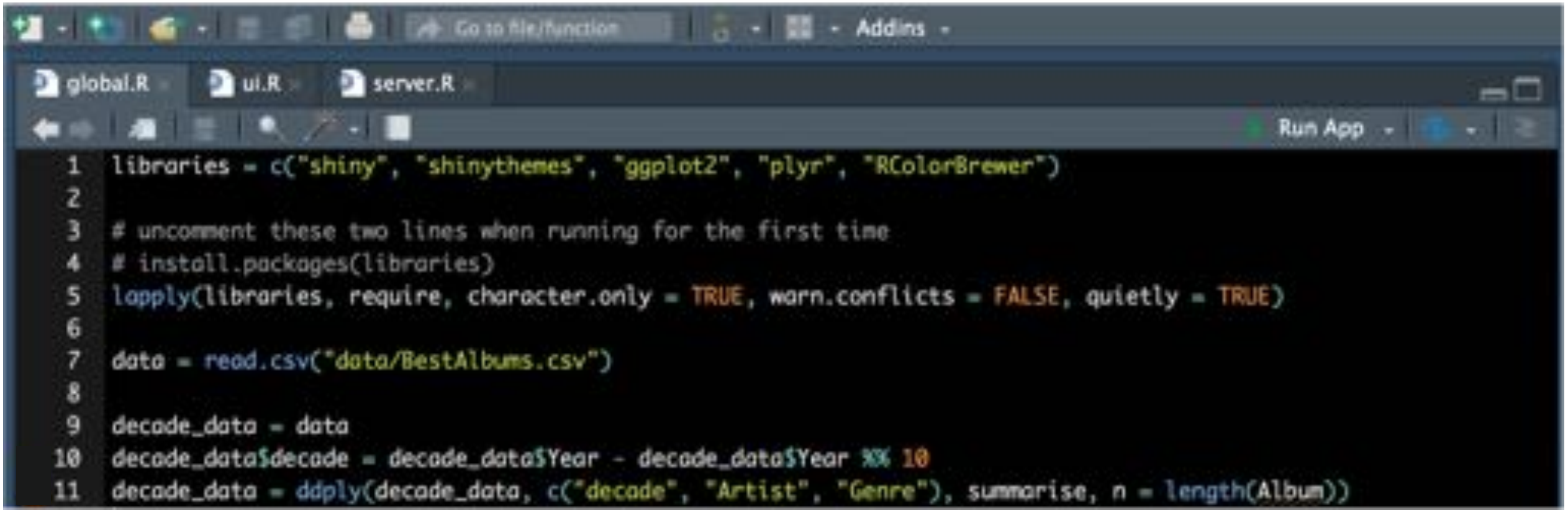


Folder/File structure

- There are 2 different ways to organize your shiny app (single file vs multiple files)
- **We will be using the following:**



Get ready for our demo



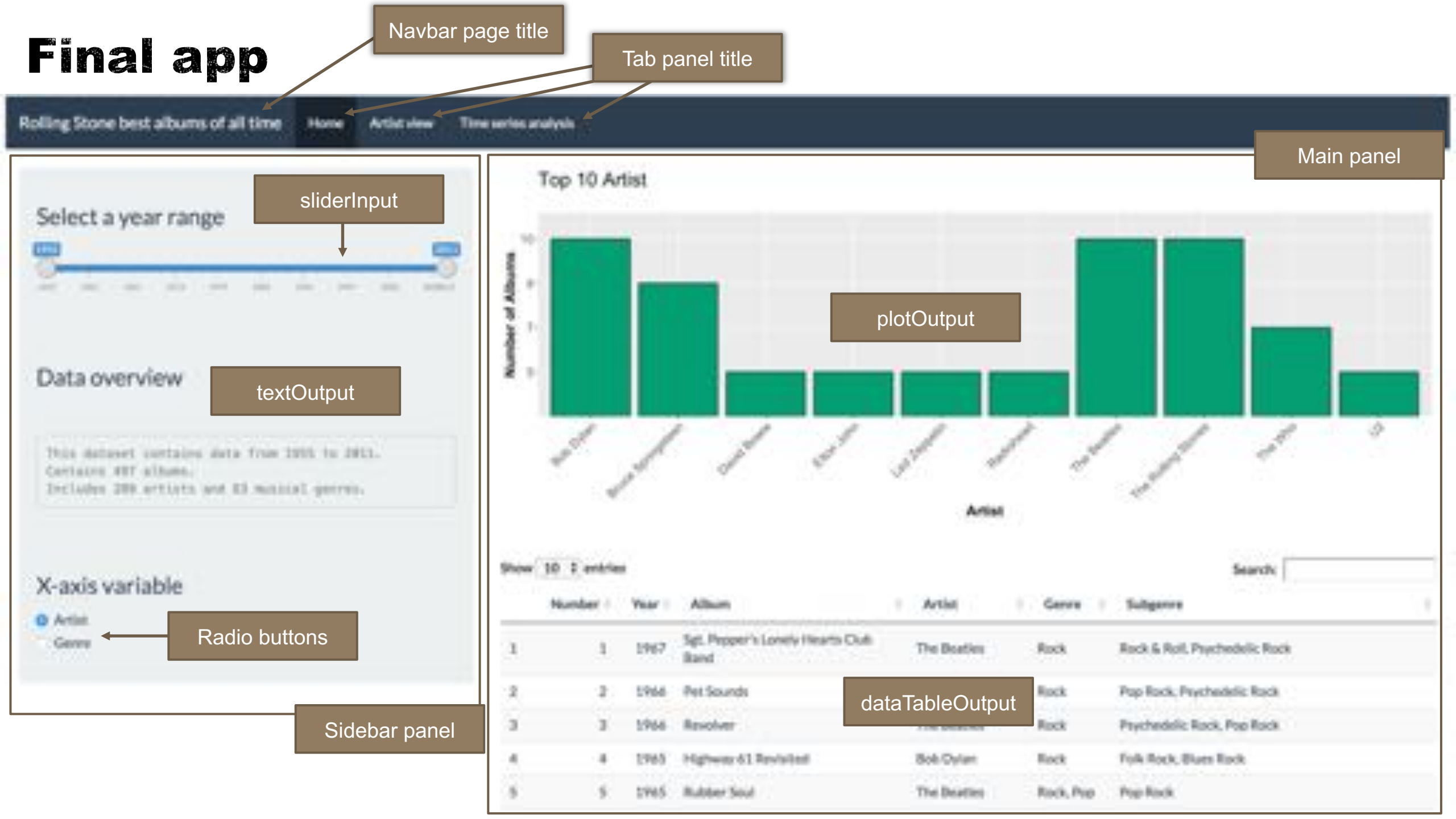
Repo available here: **https://github.com/royalmail-datascience/Meetup1_DataVisualisationShiny.git**

Write global.R

- This file is read before launching the app
  - → After launching the app, both the ui.R and server.R will have access to all objects and any of the global variables that were defined
  - → **global.R** needs to be sourced in ui.R and server.R

DEMO

Final app


Navbar page title
Tab panel title
Rolling Stone best albums of all time
Home
Artist view
Time series analysis
Main panel
sliderInput
Select a year range
Data overview
textOutput
This dataset contains data from 1955 to 2011.
Contains 497 albums.
Includes 209 artists and 13 musical genres.
X-axis variable
Artist
Genre
Radio buttons
Sidebar panel
Top 10 Artist
plotOutput
Number of Albums
Artist
Show 10 entries
Search:
Number
Year
Album
Artist
Genre
Subgenre
1 1 1967 Sgt. Pepper's Lonely Hearts Club Band The Beatles Rock Rock & Roll, Psychedelic Rock
2 2 1966 Pet Sounds Rock Pop Rock, Psychedelic Rock
3 3 1966 Revolver Rock Psychedelic Rock, Pop Rock
4 4 1965 Highway 61 Revisited Bob Dylan Rock Folk Rock, Blues Rock
5 5 1965 Rubber Soul The Beatles Rock, Pop Pop Rock
dataTableOutput

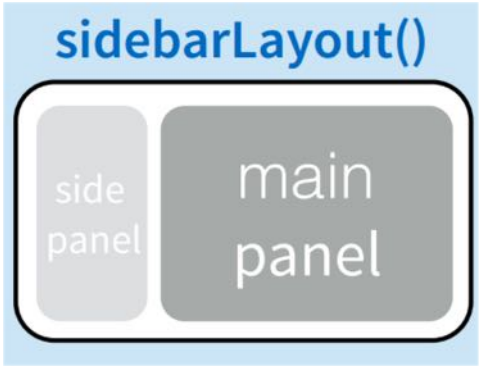
Write ui.R

- **Layout**

UI Layout

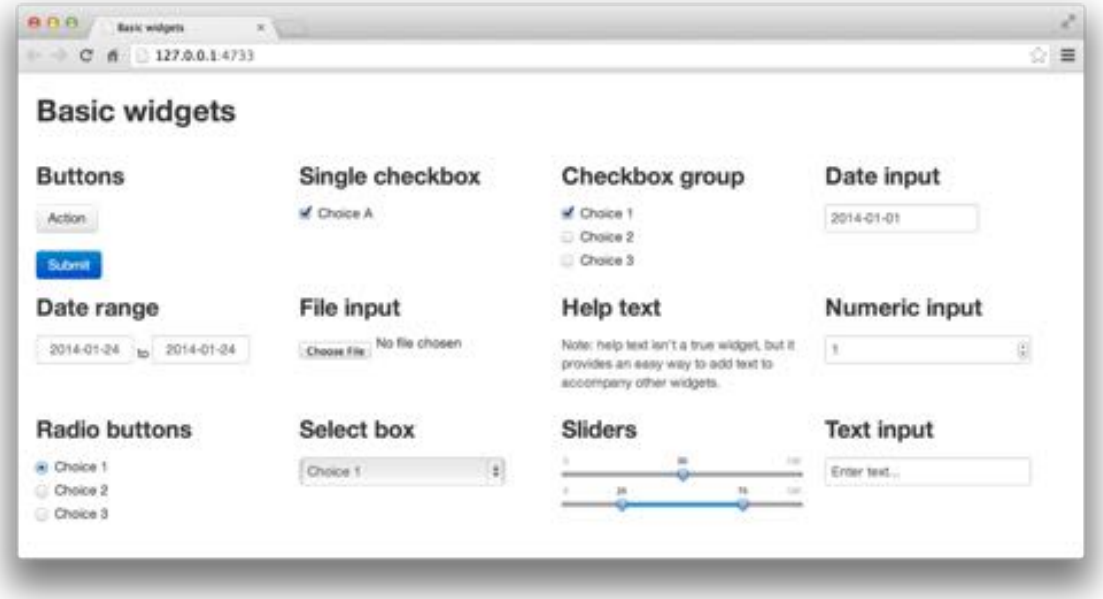
Functions for laying out the user interface for your application.

`fluidPage (fluidRow)`	Create a page with fluid layout
`navbarPage (navbarMenu)`	Create a page with a top level navigation bar
`tabPanel`	Create a tab panel
`sidebarLayout`	Layout a sidebar and main area
`sidebarPanel`	Create a sidebar panel
`mainPanel`	Create a main panel



- **Input**

```
sliderInput(inputId = “myid”,
label = “this will show in the
UI”, …)
```



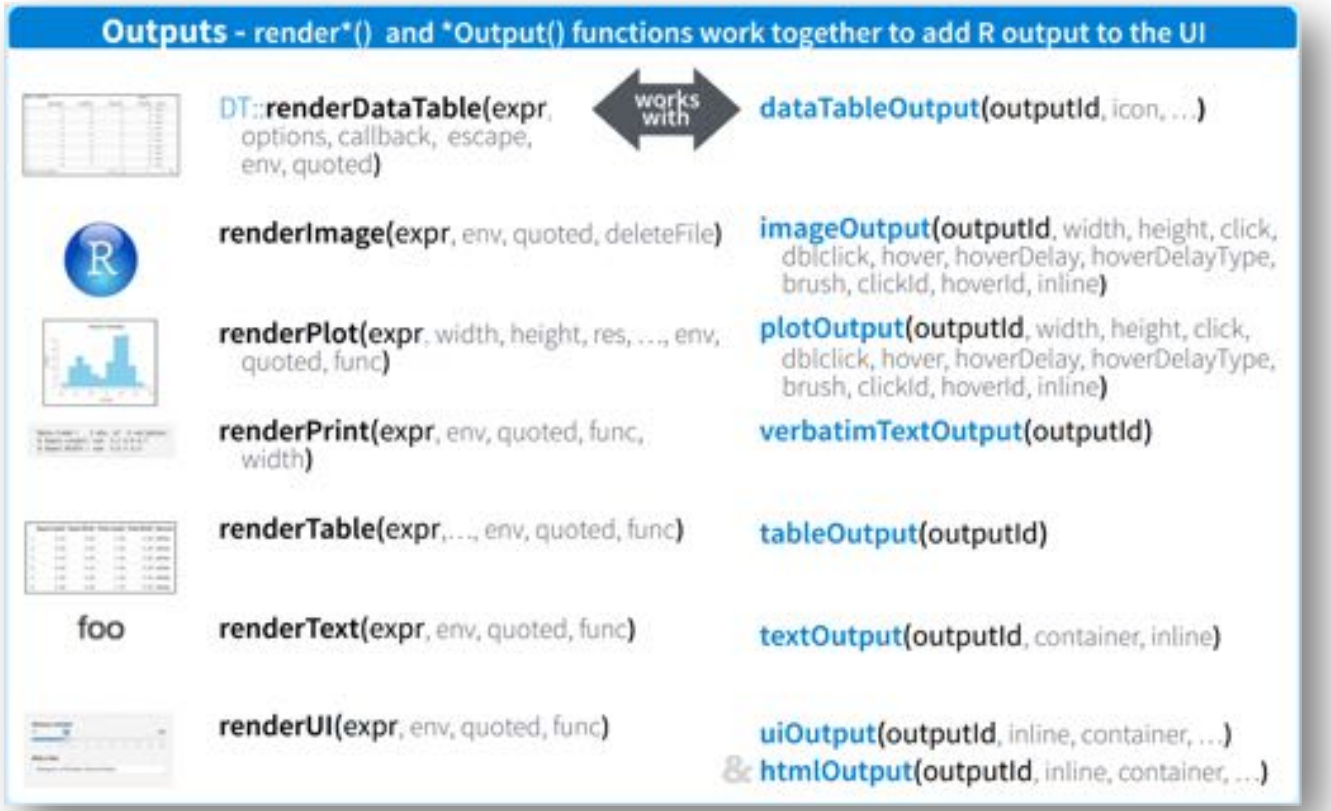
- **Output**

```
plotOutput(“histogram”)
```

DEMO

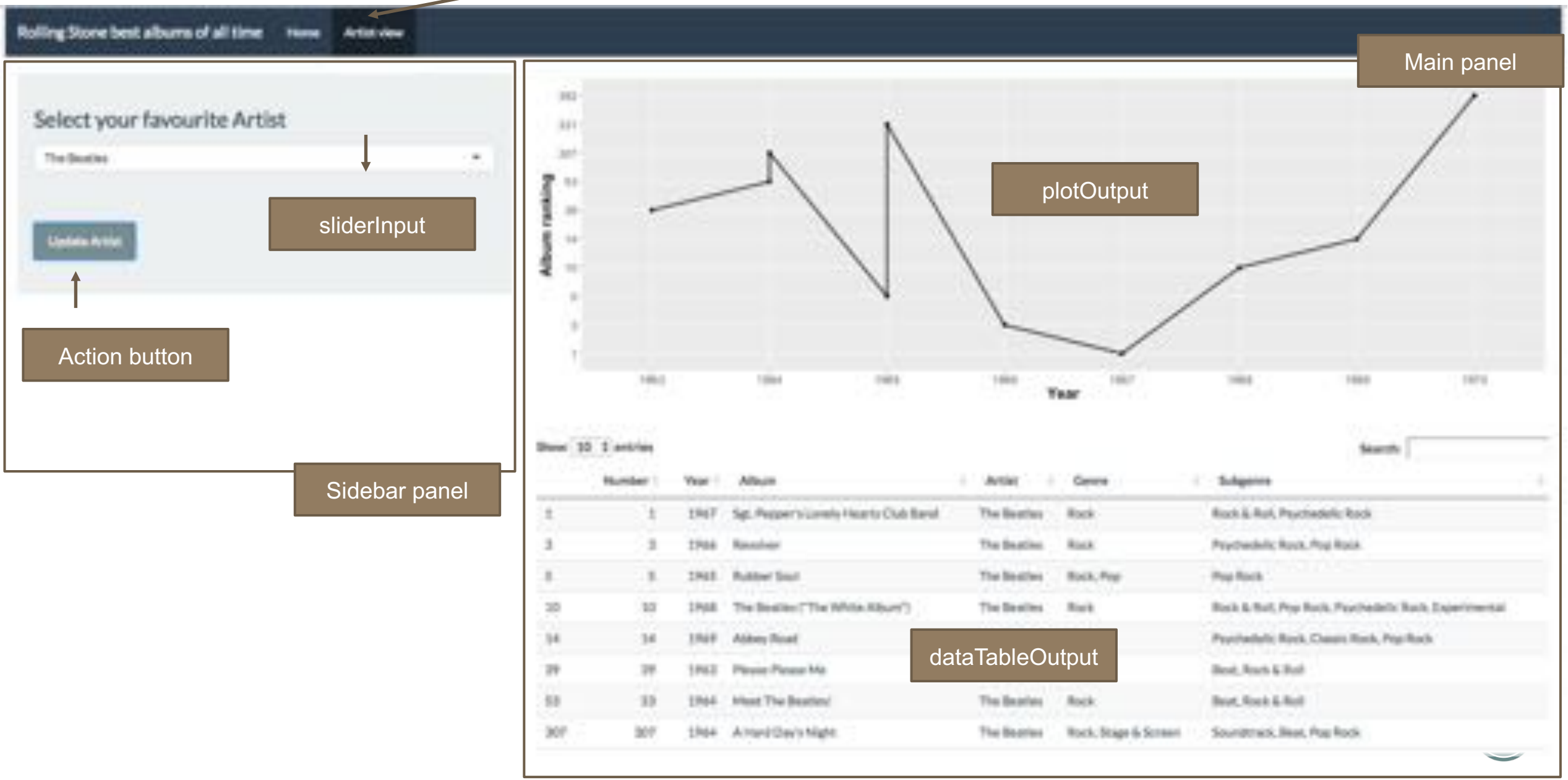
Write server.R

- Render output
- Data manipulation

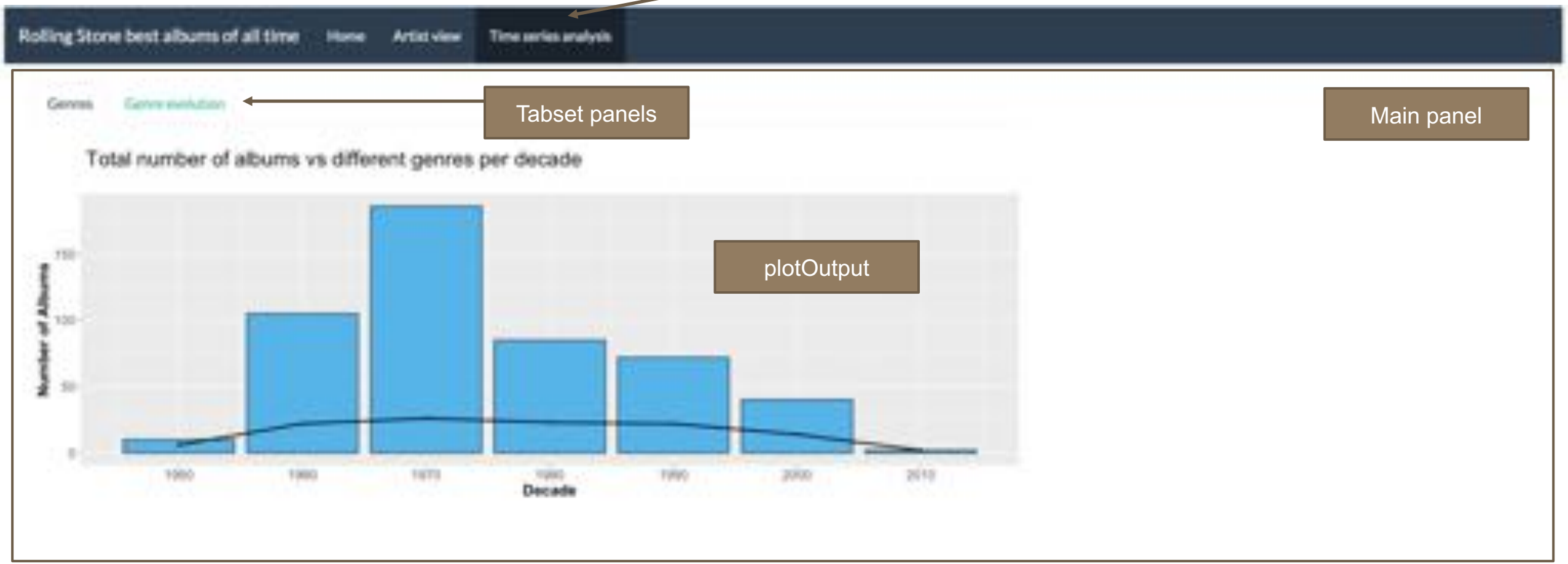


DEMO

Final app


Tab panel title
Rolling Stone best albums of all time
Home
Artist view
Select your favourite Artist
The Beatles
Update Artist
sliderInput
Action button
Sidebar panel
Main panel
plotOutput
Album ranking
Year
Show 10 1 entries
Search:
Number
Year
Album
Artist
Genre
Subgenre
dataTableOutput

Final app


Tab panel title
Rolling Stone best albums of all time
Home
Artist view
Time series analysis
Genres
Tabset panels
Main panel
Total number of albums vs different genres per decade
plotOutput
Number of Albums
Decade

Useful resources

- http://rstudio.github.io/shinythemes/
- http://rstudio.github.io/leaflet/shiny.html
- http://shiny.rstudio.com/reference/shiny/1.1.0/
- https://github.com/rstudio/shiny-examples
- https://rstudio.github.io/shinydashboard/
- https://shiny.rstudio.com/tutorial/
- http://rstudio.github.io/shiny/tutorial/
- http://shiny.rstudio.com/articles/html-tags.html
- https://shiny.rstudio.com/images/shiny-cheatsheet.pdf
- https://shiny.rstudio.com/articles/layout-guide.html
- https://shiny.rstudio.com/gallery/
- https://www.showmeshiny.com/
- https://plot.ly/r/shiny-gallery/
- https://shinyapps-recent.appspot.com/recent.html
- https://deanattali.com/blog/building-shiny-apps-tutorial/
- https://bookdown.org/weicheng/shinyTutorial/resources.html
- http://stcorp.nl/R_course/tutorial_shiny.html
- http://zevross.com/blog/2016/04/19/r-powered-web-applications-with-shiny-a-tutorial-and-cheat-sheet-with-40-example-apps/
- https://www.rstudio.com/resources/webinars/shiny-developer-conference/

Exercise

Repo available here: **https://github.com/royalmail-datascience/Meetup1_DataVisualisationShiny.git**

Exercise

- Now that you have your shiny app up and running try to play with it by doing the following tasks (solutions can be found in the repository)

  1. Add the **Year** option on *"Home"* panel → X-axis (and update the plot accordingly)

  2. Add two new tabsets on *"Time series analysis" for* **Artist** and **Artist evolution** (use Genre tabs as templates)

Exercise 1 solution

1. Add the **Year** option on *"Home"* panel → X-axis (and update the plot accordingly)

```
tabPanel(title = "Home", value = "home",
         sidebarLayout(
           sidebarPanel(
             sliderInput("daterange", label = h3("Select a year range"),
                         min = min(data$Year), max = max(data$Year),
                         value = c(min(data$Year), max(data$Year)), sep = "
             br(), br(),
             h3(textOutput("header")),
             br(), br(),
             textOutput("summary"),
             br(), br(),
             radioButtons("checkGroup", label = h3("X-axis variable"),
                                choices = list("Artist" = 1, "Genre" = 2, "Year" = 3),
                                selected = 1)
```

```
output$topalbums = renderPlot({
  datatoplot = datatoplot()

  variable = input$checkGroup
  if (variable == 1){
    variable_name = "Artist"
  } else if (variable == 2){
    variable_name = "Genre"
  } else if (variable == 3){
    variable_name = "Year"
  }
```

Exercise 2 solution

2. Add two new tabs on templates)

```
tabPanel(title = "Time serie
        mainPanel(
          tabsetPanel(
            tabPanel("Genre
                  plotOutput
            tabPanel("Genre
                  plotOutput
            tabPanel("Artis
                        plotOu
            tabPanel("Artis
                        plotOu
          ) #tabsetpanel
```

```
output$artists = renderPlot({
  datatoplot = ddply(decade_data, c("decade", "Artist"), summarise, n = sum(n))
  datatoplot_artist = ddply(datatoplot, c("decade"), summarise, n = length(Artist))
  datatoplot_total = ddply(decade_data, c("decade"), summarise, total = sum(n))
  datatoplot_mix = merge(datatoplot_artist, datatoplot_total, by = "decade")

  ggplot(datatoplot_mix) +
    geom_bar(aes(decade, total), stat = "identity", fill = "#5684E9", colour = "black") +
    geom_line(aes(decade, n), size = 1) +
    geom_point(aes(decade, n), size = 1) +
    xlab("Decade") + ylab("Number of Albums") +
    ggtitle("\n Total number of artists vs different artists per decade \n") +
    scale_x_continuous(breaks = seq(1950, 2010, 10)) +
    theme(axis.text = element_text(size = 14), axis.title = element_text(size = 16, face = "bold"),
          title = element_text(size = 18))
})

output$yearsartist = renderPlot({
  datatoplot = ddply(decade_data, c("decade", "Artist"), summarise, n = sum(n))
  datatoplot = subset(datatoplot, n > 1)

  ggplot(datatoplot, aes(Artist, decade)) + geom_tile(aes(fill = as.factor(n)), colour = "black") +
    scale_y_continuous(breaks = seq(1950, 2010, by = 10)) +
    xlab("Artist") + ylab("Decade") +
    scale_fill_manual(name = "Number of Albums", values = brewer.pal(n = 6, name = "YlGnBu")) +
    theme(axis.text = element_text(size = 14), axis.title = element_text(size = 16, face = "bold"),
          legend.title = element_text(size = 16, face = "bold"), legend.text = element_text(size = 16),
          axis.text.x = element_text(angle = 45, hjust = 1))
})
```

Thanks!

Slides and solution to exercises will be made available on the repo after the workshop

ana.guedes@royalmail.com for any questions!