

# Shiny R package

---

Maide Ünal & Alexander Schendelmann



# Example

Lets go to New York City again!

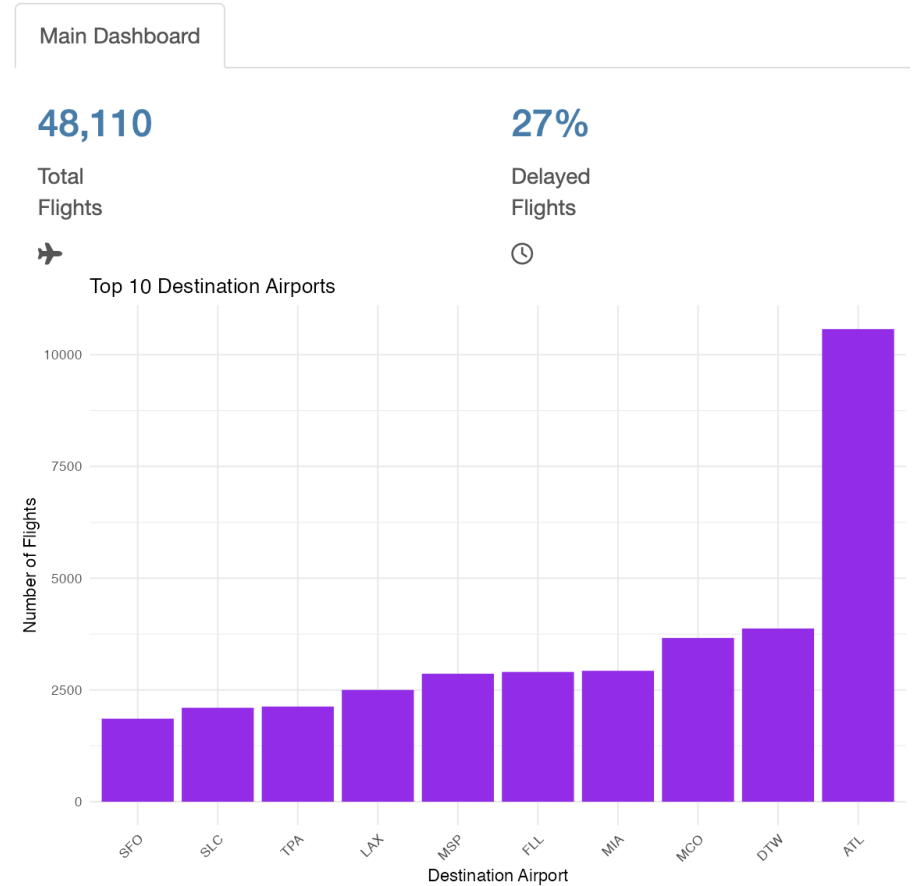
## Flights - Dashboard

**Airline**

Delta Air Lines Inc. ▼

**Time period**

- January
- February
- March
- April
- May
- June
- July
- August
- September
- October
- November
- December
- All Year



# Overview

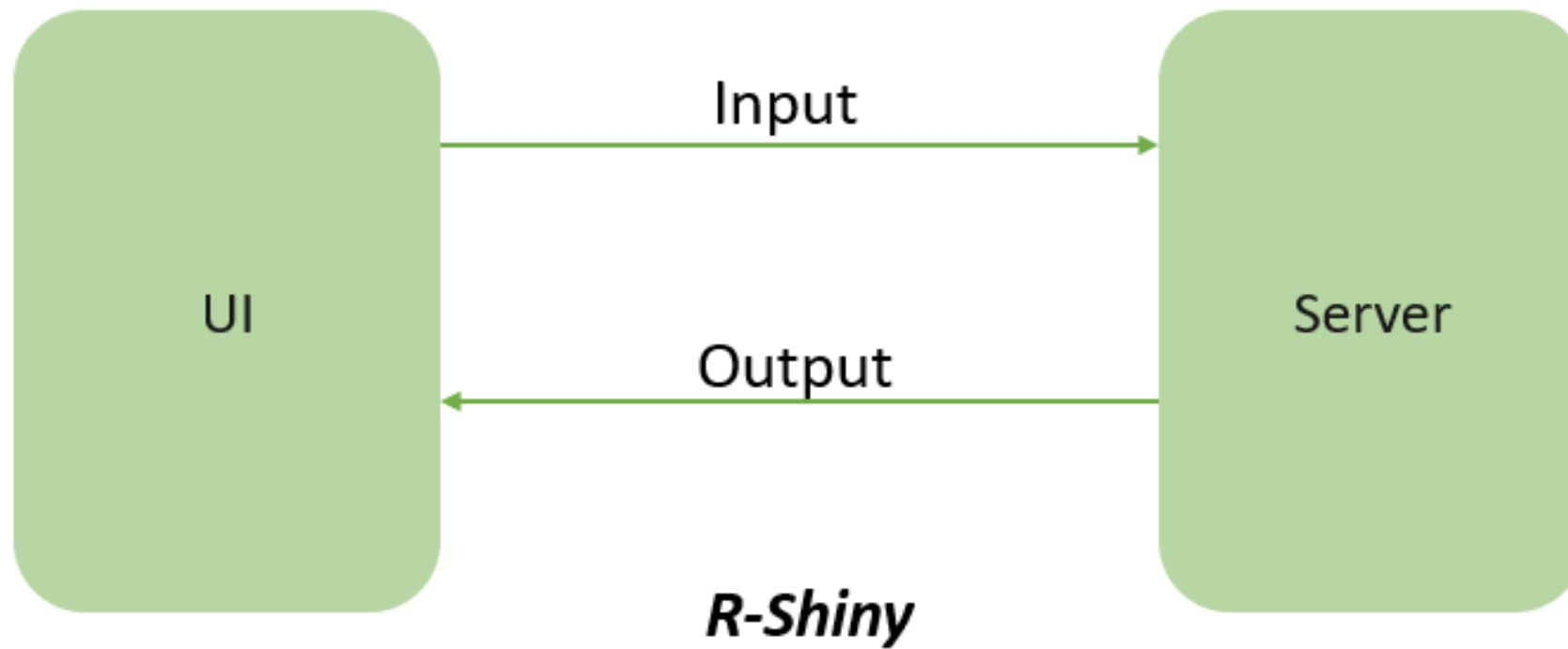
- What is Shiny?
- User Interface
- Server
- Sharing your app
- Why Shiny?

# What is Shiny?

- Open-source Web application framework for R
- Stylish, interactive, data-driven web apps
- Perfect to share findings in your dataset
- Browser-based → no need of R for viewing

# Shiny project structure

- Components of a Shiny App
  - User Interface – layout and appearance of the web app
  - Server – handles user inputs, processes data, and generates outputs dynamically
- Call of the shinyApp() function
  - Fuses UI and server



# User Interface (UI)

# User Interface

- Frontend that accepts user input values
- Layout and Appearance
  - Design and arrangement of visual elements
- Widgets for Input
  - Text input fields, dropdown menus
- User Interaction
  - Web elements allowing user engagement
  - Mechanisms to send user inputs to the Shiny App
- Outputs
  - Visual Representations
  - Graphs and plots

# Why do web layouts matter in Shiny?

- Web application framework
  - Relies on HTML, CSS, JavaScript for its structure, looks, and functionality



# Layout

- Create more visually appealing apps -> **layout functions**
- Bootstrap layouts - popular HTML/CSS framework
- Customize layout with R's bslib package
- Alternatives
  - shinydashboard (create dashboards)
  - shinyMobile (optimal for mobile apps)
  - shiny.semantic (semantic UI integration)
  - shinymaterial (material design components)

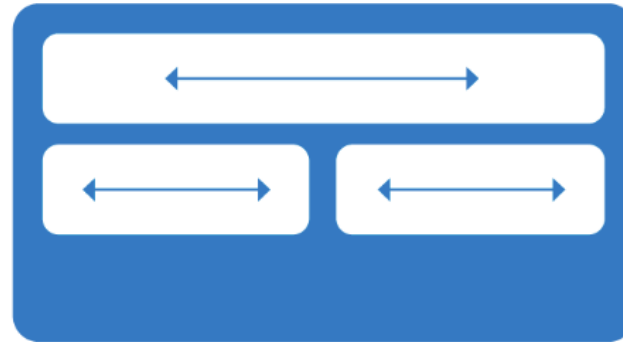
# Filling layout

- **Page\_fixed()**
  - Places components into a grid with a fixed width
- **Page\_fluid()**
  - Places components into a grid that always occupies the full width
- **Page\_fillable()**
  - Places components into a grid that always occupies the full width and height

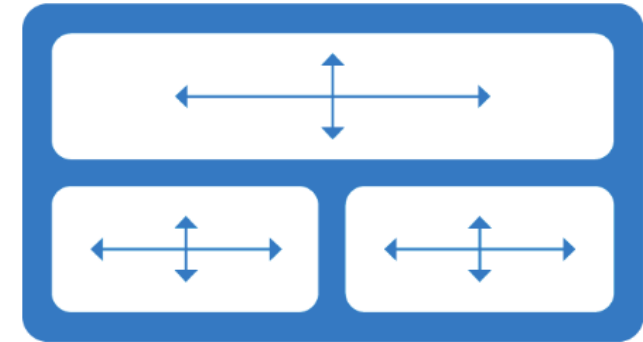
`page_fixed()`



`page_fluid()`



`page_fillable()`



# fluidPage()

```
ui <- fluidPage(  
  titlePanel(),  
  sidebarLayout(  
    sidebarPanel(),  
    mainPanel()  
  )  
)  
server <- function(input, output) { }  
shinyApp(ui, server)
```

fluidPage()

titlePanel()

sidebarLayout()

sidebarPanel()

mainPanel()

# Multi-Row

```
ui <- fluidPage(  
  fluidRow(  
    column(4,  
      ...  
    ),  
    column(8,  
      ...  
    )  
  ),  
  fluidRow(  
    column(6,  
      ...  
    ),  
    column(6,  
      ...  
    )  
  )  
)
```

fluidPage()

fluidRow()

column(4)

column(8)

fluidRow()

column(6)

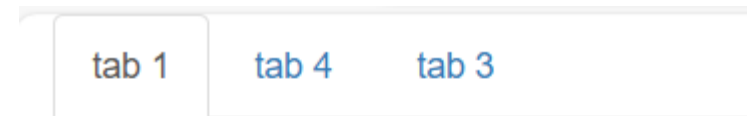
column(6)

# Multi-page layouts - tabsets

- `tabPanel()` – illusion of multiple pages
- Displayed horizontally
- Limit to how many tabs you can use, particularly for long titles

# TabsetPanel()

```
ui <- fluidPage(  
  tabsetPanel(  
    tabPanel("tab 1"),  
    tabPanel("tab 4"),  
    tabPanel("tab 3")  
  )  
)  
  
server <- function(input, output) { }  
  
shinyApp(ui, server)
```



# Multi-page layouts – navlists and navbars

- Two alternative layouts
  - `navbarPage()` and `navbarMenu()`
- More tabs with longer titles
- `NavlisPanel()`
  - Tab titles vertically in a sidebar
- `NavbarPage()` and `NavbarMenu()`
  - Tab titles horizontally and add drop-down menus

# NavlistPanel()

```
ui <- fluidPage(  
  navlistPanel(  
    id = "tabset",  
    "Heading 1",  
    tabPanel("panel 1", "Panel one contents"),  
    "Heading 2",  
    tabPanel("panel 2", "Panel two contents"),  
    tabPanel("panel 3", "Panel three contents")  
  )  
)
```

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

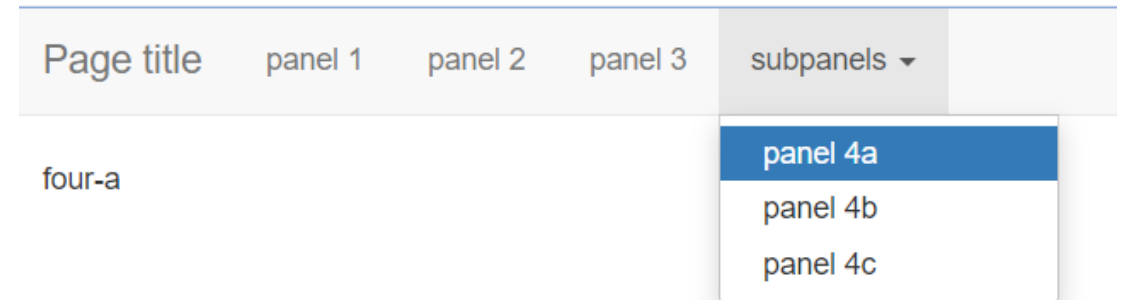


Panel one contents

# NavbarPage()

```
ui <- navbarPage(  
  "Page title",  
  tabPanel("panel 1", "one"),  
  tabPanel("panel 2", "two"),  
  tabPanel("panel 3", "three"),  
  navbarMenu("subpanels",  
    tabPanel("panel 4a", "four-a"),  
    tabPanel("panel 4b", "four-b"),  
    tabPanel("panel 4c", "four-c")  
  )  
)
```

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



# Themes

```
install.packages("shinythemes")

ui <- fluidPage(theme = shinytheme("cerulean"),
  ...
)
server <- function(input, output) { }

shinyApp(ui, server)
```

Cosmo    Navbar 1    Plot    Table

Tab 1    Tab 2    Tab 3

File input:  
Browse...    No file selected

Text input:  
general

Slider input:  
1    30    100

Default actionButton:  
Search

actionButton with CSS class:  
Action button

Table

| speed | dist  |
|-------|-------|
| 4.00  | 2.00  |
| 4.00  | 10.00 |
| 7.00  | 4.00  |
| 7.00  | 22.00 |

Verbatim text output  
general, 30, NULL

Header 1  
Header 2  
Header 3  
Header 4  
Header 5

Darkly    Navbar 1    Plot    Table

Tab 1    Tab 2    Tab 3

File input:  
Browse...    No file selected

Text input:  
general

Slider input:  
1    30    100

Default actionButton:  
Search

actionButton with CSS class:  
Action button

Table

| speed | dist  |
|-------|-------|
| 4.00  | 2.00  |
| 4.00  | 10.00 |
| 7.00  | 4.00  |
| 7.00  | 22.00 |

Verbatim text output  
general, 30, NULL

Header 1  
Header 2  
Header 3  
Header 4  
Header 5

United    Navbar 1    Plot    Table

Tab 1    Tab 2    Tab 3

File input:  
Browse...    No file selected

Text input:  
general

Slider input:  
1    30    100

Default actionButton:  
Search

actionButton with CSS class:  
Action button

Table

| speed | dist  |
|-------|-------|
| 4.00  | 2.00  |
| 4.00  | 10.00 |
| 7.00  | 4.00  |
| 7.00  | 22.00 |

Verbatim text output  
general, 30, NULL

Header 1  
Header 2  
Header 3  
Header 4  
Header 5

# Exercise 1 – Warmup

1. Open a Shiny Web App in R
2. Modify the code
  - The sidebar should be on the right instead of the left side
  - Choose and add a theme you like
3. Add two tabs
  - First one should contain the demo code
  - Second one should have title „Shiny Layouts“
  - Third one should be a drop-down menu with three dropdowns

# UI: Inputs – Common structure

- InputId
  - identifier used to connect the front end with the back end
  - UI has input with ID „name“, server function will access it with `input$name`
- Label
  - Human-readable label for the control
- Value
  - Set default value if possible

# UI: Inputs – Free text

```
ui <- fluidPage(  
  textInput("name", "What's your name?"),  
  passwordInput("password", "What's your password?"),  
  textAreaInput("story", "Tell me about yourself", rows = 3)  
)  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```

What's your name?

What's your password?

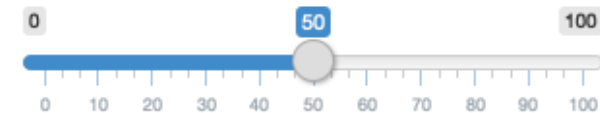
Tell me about yourself

# UI: Inputs – Numeric inputs

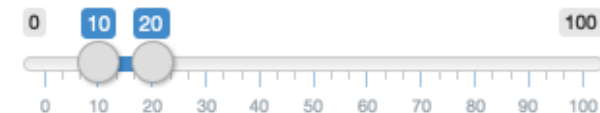
```
ui <- fluidPage(  
  numericInput("num", "Number one", value = 0, min = 0, max = 100),  
  sliderInput("num2", "Number two", value = 50, min = 0, max = 100),  
  sliderInput("rng", "Range", value = c(10, 20), min = 0, max = 100)  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```

Number one

Number two



Range



# UI: Inputs – Dates

```
ui <- fluidPage(
  dateInput("dob", "When were you born?"),
  dateRangeInput("holiday", "When do you want to go on vacation next?")
)

server <- function(input, output) {}

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

When were you born?

2020-09-16

When do you want to go on vacation next?

2020-09-16

to

2020-09-16

When were you born?

2024-05-26

« May 2024 »

| Su | Mo | Tu | We | Th | Fr | Sa |
|----|----|----|----|----|----|----|
| 28 | 29 | 30 | 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  | 9  | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | 1  |
| 2  | 3  | 4  | 5  | 6  | 7  | 8  |

# UI: Inputs – Limited Choice

```
animals <- c("dog", "cat", "mouse", "bird", "other", "I hate animals")
ui <- fluidPage(
  selectInput("state", "What's your favourite state?", state.name),
  radioButtons("animal", "What's your favourite animal?", animals)
)

server <- function(input, output) {}

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

What's your favourite state?

Alabama

What's your favourite animal?

- dog
- cat
- mouse
- bird
- other
- I hate animals

# UI: Inputs – Limited Choice

```
ui <- fluidPage(  
  selectInput(  
    "state", "What's your favourite state?", state.name,  
    multiple = TRUE  
  )  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```

What's your favourite state?

Texas Cal

California

# UI: Inputs – Limited Choice

## checkboxInputGroupInput

```
ui <- fluidPage(  
  checkboxInputGroupInput("animal", "What animals do you like?", animals)  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```

What animals do you like?

- dog
- cat
- mouse
- bird
- other
- I hate animals

## checkboxInput

```
ui <- fluidPage(  
  checkboxInput("cleanup", "Clean up?", value = TRUE),  
  checkboxInput("shutdown", "Shutdown?")  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```

- Clean up?
- Shutdown?

# UI: Inputs – File uploads

```
ui <- fluidPage(  
  fileInput("upload", NULL)  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```



# UI: Inputs – Action buttons

```
ui <- fluidPage(  
  actionButton("click", "Click me!"),  
  actionButton("drink", "Drink me!", icon = icon("cocktail"))  
)  
  
server <- function(input, output) { }  
  
shinyApp(ui, server)
```

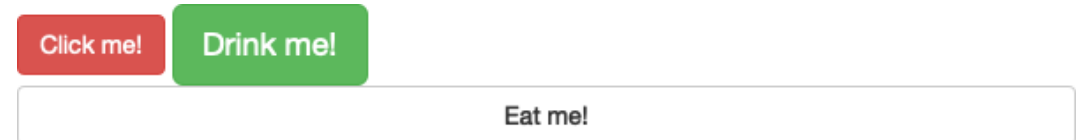


<https://bootstrapdocs.com/v3.3.6/docs/css/#buttons>

# UI: Inputs – Action buttons

```
ui <- fluidPage(  
  fluidRow(  
    actionButton("click", "Click me!", class = "btn-danger"),  
    actionButton("drink", "Drink me!", class = "btn-lg btn-success")  
  ),  
  fluidRow(  
    actionButton("eat", "Eat me!", class = "btn-block")  
  )  
)
```

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



# Exercise 1 – UI

1. Use "cerulean" as our theme and create a title
2. Add a sidebar with two drop-down menus:
  - a) choice of the airline
  - b) choice of the time period (month)
3. Create a main panel with a blank page called "Main Dashboard"

# Server

# Lets bring our Dashboard to life!

- Reactive output: Whenever the user toggles a widget, the UI responds
- Logic is implemented in the server using render functions
- The UI shows the requested outputs using output functions that turn R objects into UI output

# Reactivity idea

Spreadsheet

File Edit View Insert Format Data Tools Add-ons H

100% \$ % .0 .00 123

fx =B3\*2

|    | A | B   | C     | D |
|----|---|-----|-------|---|
| 1  |   |     |       |   |
| 2  |   |     |       |   |
| 3  |   | 100 | =B3*2 |   |
| 4  |   |     |       |   |
| 5  |   |     |       |   |
| 6  |   |     |       |   |
| 7  |   |     |       |   |
| 8  |   |     |       |   |
| 9  |   |     |       |   |
| 10 |   |     |       |   |
| 11 |   |     |       |   |



Spreadsheet

File Edit View Insert Format Data Tools Add-ons H

100% \$ % .0 .00 123 Aria

fx =B3\*2

|    | A | B   | C   | D |
|----|---|-----|-----|---|
| 1  |   |     |     |   |
| 2  |   |     |     |   |
| 3  |   | 300 | 600 |   |
| 4  |   |     |     |   |
| 5  |   |     |     |   |
| 6  |   |     |     |   |
| 7  |   |     |     |   |
| 8  |   |     |     |   |
| 9  |   |     |     |   |
| 10 |   |     |     |   |
| 11 |   |     |     |   |

# Reactive flow in Shiny (simplified)



# Some output and render functions

| Output function (UI)         | Render function (Server)     | Object created                         |
|------------------------------|------------------------------|--|
| <code>dataTableOutput</code> | <code>renderDataTable</code> | <code>DataTable</code>                 |
| <code>imageOutput</code>     | <code>renderImage</code>     | <code>image</code>                     |
| <code>plotOutput</code>      | <code>renderPlot</code>      | <code>plot</code>                      |
| <code>textOutput</code>      | <code>renderText</code>      | <code>text</code>                      |
| <code>valueBoxOutput</code>  | <code>renderValueBox</code>  | <code>valueBox</code> (Shiny specific) |
| <code>uiOutput</code>        | <code>renderUI</code>        | <code>HTML</code>                      |

# Example: Show selected input choice

```
ui <- page_sidebar(  
  title = "My reactive Dashboard",  
  sidebar = sidebar(  
    selectInput(  
      "var",  
      label = "Choose a variable to display",  
      choices = c("Choice 1",  
                  "Choice 2"),  
      selected = "Choice 1"  
    )  
  ),  
  textOutput("selected_var")  
)
```

```
server <- function(input, output) {  
  output$selected_var <- renderText({  
    input$var  
  })  
}
```

## Server syntax:

- Access input/output variables using the \$ operator
- Surround R expressions by {} in render\* functions

# Example: Show plots

```
ui <- page_sidebar(  
  title = "My reactive Dashboard",  
  sidebar = sidebar(  
    selectInput(  
      "var",  
      label = "Choose a variable to display",  
      choices = c("Choice 1",  
                  "Choice 2"),  
      selected = "Choice 1"  
    )  
  ),  
  plotOutput("myPlot")  
)
```

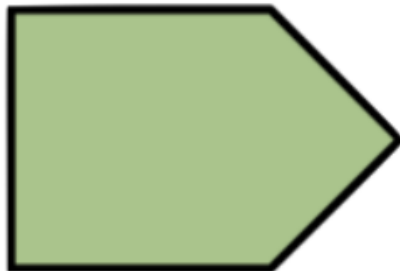
```
server <- function(input, output) {  
  output$myPlot <- renderPlot({  
    data <- data.frame(x = 1:10, y = rnorm(10))  
    ggplot(data, aes(x = x, y = y)) +  
      geom_point() +  
      labs(title = "Sample Plot")  
  })  
}
```

## Note:

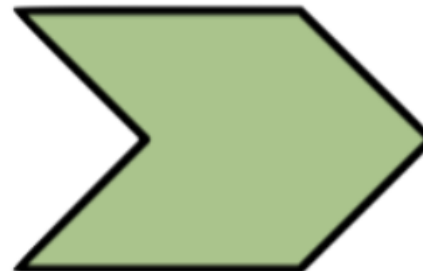
- regular use of ggplot()
- This plot does not depend on the input choice

# Reactive elements

Reactive  
input



Reactive  
expression



Reactive  
output



# Reactive expressions

- Expression knows if it is out of date (if not, it is not recomputed)  
→ “lazy” in contrast to regular functions
- Idea: Only compute updates when necessary
  - Imagine a complex app with multiple inputs
  - Change of one input should only lead to certain recomputations
  - Recomputing everything leads to slowdowns
- Readability is improved: decompose complex calculations, avoid copy-and-paste when used in multiple places

# Example: Reactive Expressions

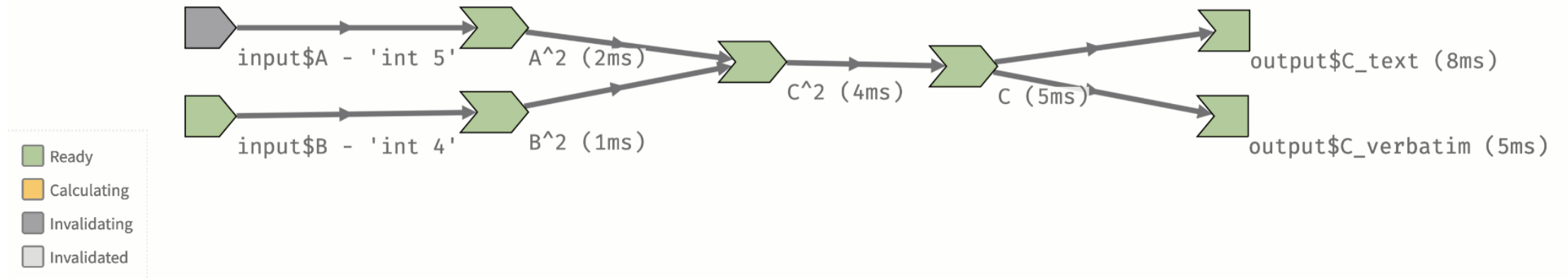
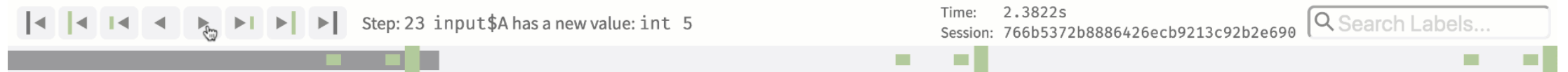
```
ui <- page_sidebar(  
  title = "My reactive Dashboard",  
  sidebar = sidebar(  
    selectInput(  
      "var",  
      label = "Choose a variable to display",  
      choices = c("Choice 1",  
                  "Choice 2"),  
      selected = "Choice 1"  
    )  
  ),  
  plotOutput("myPlot")  
)
```

```
server <- function(input, output) {  
  data <- reactive({  
    if (input$var == "Choice 1")  
      data <- data.frame(x = 1:10, y = rnorm(10))  
    else  
      data <- data.frame(x = 1:100, y = rnorm(100))  
  })  
  
  output$myPlot <- renderPlot({  
    ggplot(data()) aes(x = x, y = y) +  
      geom_point() +  
      labs(title = "Sample Plot")  
  })  
}
```

# Reactlog

- Dependency structure may become complex
- Overview provided by the reactlog package
- Setup:
  - Installation via `install.packages("reactlog")`
  - Run `options(shiny.reactlog = TRUE)` command
  - Launch app
  - press **Ctrl + F3** (or on Mac: **Cmd + F3**)

# Example Reactlog



## Exercise 2 – Server

1. Reactively filter the flights dataset based on the selected airline and month
2. Based on the user selections, show the following statistics:
  - a) Total number of flights
  - b) Percentage of flights with a delay of more than 5 minutes
3. Create a bar plot for the top 10 airports (reactive to selections)

# Share your app

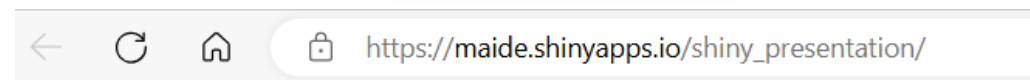
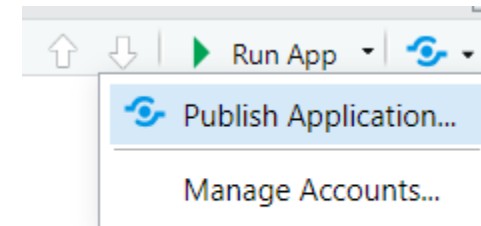
- R scripts
  - User also needs R
- Web page
  - User friendly
  - Navigate to the app through the internet with a web browser
  - Fully rendered, up to date, and ready to go

# Share as a web page

- Host shiny app yourself
- Shinyapps.io
- Shiny Server
- Posit Connect

# Deploy on Shinyapps.io

- Upload app straight from your R session to a server hosted by Posit
- Complete control over app including server administration tools
- Install rsconnect package
- Authorize Account
- Ready to deploy



Flights - Dashboard

# Why Shiny?

- Open-source language
- Full web framework
- Wide array of inputs
- Versatile when it comes to visual styling
- Powerful, customizable, and interactive dashboards that look great and respond to user input
- For anything more complex, R Shiny overthrows simple tools

# Thank you for your attention!

Questions?

# Sources

Shiny tutorial:

<https://shiny.posit.co/r/getstarted/shiny-basics/lesson4/>

<https://mastering-shiny.org/basic-ui.html>

<https://www.appsilon.com/post/shiny-application-layouts>

[https://bookdown.org/paul/shiny\\_workshop/03-user-interface.html](https://bookdown.org/paul/shiny_workshop/03-user-interface.html)

<https://rstudio.github.io/shinythemes/>

Dashboard idea:

<https://github.com/dsciencelabs/Flights-Dashboard>