

R-Shiny Workshop

COP of Biometrics, Bioinformatics and Data Management
Amsterdam - Sep. 24-29, 2018



Omar Benites Alfaro



o.benites@cgiar.org



<https://github.com/omarbenites>

Ivan Perez Masias



i.perez@cgiar.org



<https://github.com/ivanperezma>

Outline

Part I

- **Introduction**
- CIP examples
- Shiny/RStudio
- Structure
- Setup of shiny app
- Inputs & outputs
- `render*()`
- **Exercise 1**

Part II

- Render UI
- Conditional panel
- Reactivity expression
- Plot output
- **Exercise 2**
- Html output
- DT library
- Action button and observe-Event
- `rhandsondtable`
- **Exercise 3**

Part III

Shiny Dashboard

- Basics
- Structure overview
- Header
- Sidebar
- Body
- Layouts
- **Exercise 4**

Part IV

How to customize appearance

- Skins
- CSS
- Long titles
- Sidebar width
- Icons
- Statuses and colors
- **Exercise 5**

Part V

Sharing apps

- Shinyapps.io
- Shiny servers

Part I

Introduction

Objectives

The main objective of the Shiny-Workshop is to build interactive applications using Shiny and R code based on real case studies.

Learn how to use and manipulate different types of inputs and outputs in the Shiny environment.

Pre-requisites

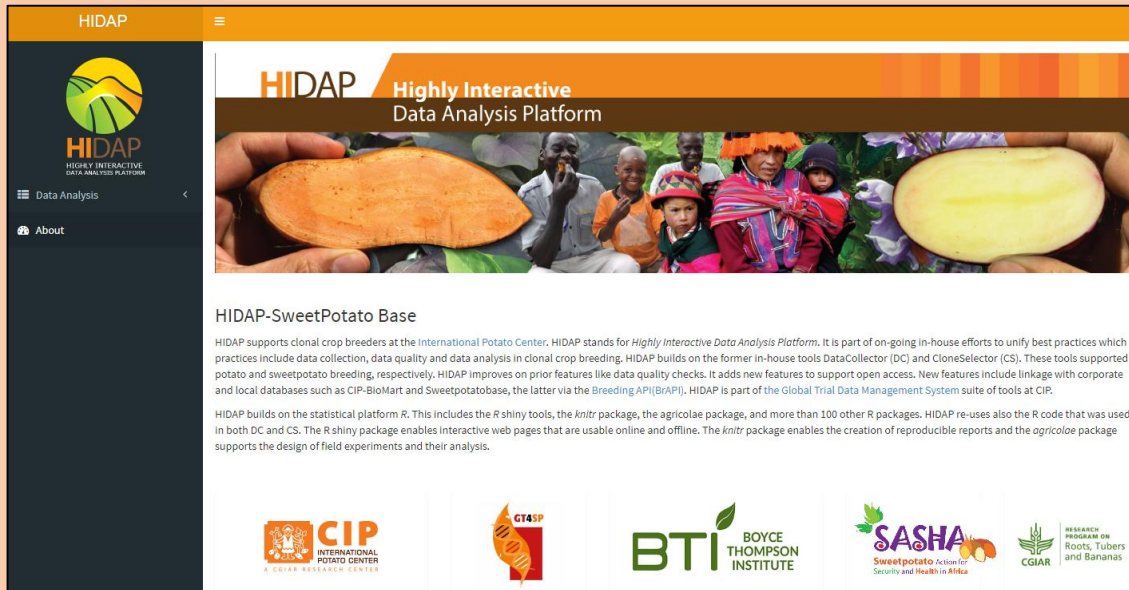
- Programming in R.
- **No previous knowledge** in languages such as HTML, PHP or Javascript is required. It only uses R code.



CIP examples

Demos of shiny.

CIP Breeding, Shiny and BRAPI and CGIAR Big Data Platform (AGROFIMS)



HIDAP Highly Interactive Data Analysis Platform

HIDAP-SweetPotato Base

HIDAP supports clonal crop breeders at the International Potato Center. HIDAP stands for *Highly Interactive Data Analysis Platform*. It is part of on-going in-house efforts to unify best practices which practices include data collection, data quality and data analysis in clonal crop breeding. HIDAP builds on the former in-house tools DataCollector (DC) and CloneSelector (CS). These tools supported potato and sweetpotato breeding, respectively. HIDAP improves on prior features like data quality checks. It adds new features to support open access. New features include linkage with corporate and local databases such as CIP-BioMart and Sweetpotatobase, the latter via the *Breeding API (BRAPI)*. HIDAP is part of the *Global Trial Data Management System* suite of tools at CIP.

HIDAP builds on the statistical platform R. This includes the R shiny tools, the *knitr* package, the *agricolae* package, and more than 100 other R packages. HIDAP re-uses also the R code that was used in both DC and CS. The R shiny package enables interactive web pages that are usable online and offline. The *knitr* package enables the creation of reproducible reports and the *agricolae* package supports the design of field experiments and their analysis.

CIP INTERNATIONAL POTATO CENTER
GT4SP
BTI BOYCE THOMPSON INSTITUTE
SASHA Sweetpotato Action for Security and Health in Africa
CGIAR RESEARCH PROGRAM ON ROOTS, TUBERS and Bananas

<https://research.cip.cgiar.org/gtdms/>
https://apps.cipotato.org/hidap_sbase/



HIDAP AgroFIMS Agronomy Field Management System

HIDAP AgroFIMS v0.2.0

The Agronomy Field Information Management System (AgroFIMS) has been developed on CGIAR's HIDAP (highly-interactive Data Analysis Platform) created by CGIAR's International Potato Center, CIP. AgroFIMS draws fully on ontologies, particularly the Agronomy Crop Ontology. It consists of modules that represent the typical cycle of operations in agronomic trial management, and enables the creation of data collection sheets using the same ontology-based set of variables, terminology, units and protocols. AgroFIMS then:

- Standardizes data collection and description for easy aggregation and inter-linking across disparate datasets;
- Allows easy integration with HIDAP breeding data, or any other ontology-based datasets;
- Functions as a data staging repository, allowing data uploads with view/edit permissions;
- Enables data quality checks, statistical analysis of the data collected, and the generation of sophisticated statistics reports;
- Aligns a priori with CGIAR's CIP Core metadata schema;
- Enables easy upload to the institutional repositories, and much more.

Funding for AgroFIMS was provided by the Bill and Melinda Gates Foundation's Open Access, Open Data Initiative, and the CGIAR Big Data Platform.

CGIAR Platform for Big Data in Agriculture
CIAT International Center for Tropical Agriculture
CIP INTERNATIONAL POTATO CENTER
IFPRI INTERNATIONAL FOOD POLICY RESEARCH INSTITUTE
Bioversity INTERNATIONAL

<https://apps.cipotato.org/hidapagrofims/>

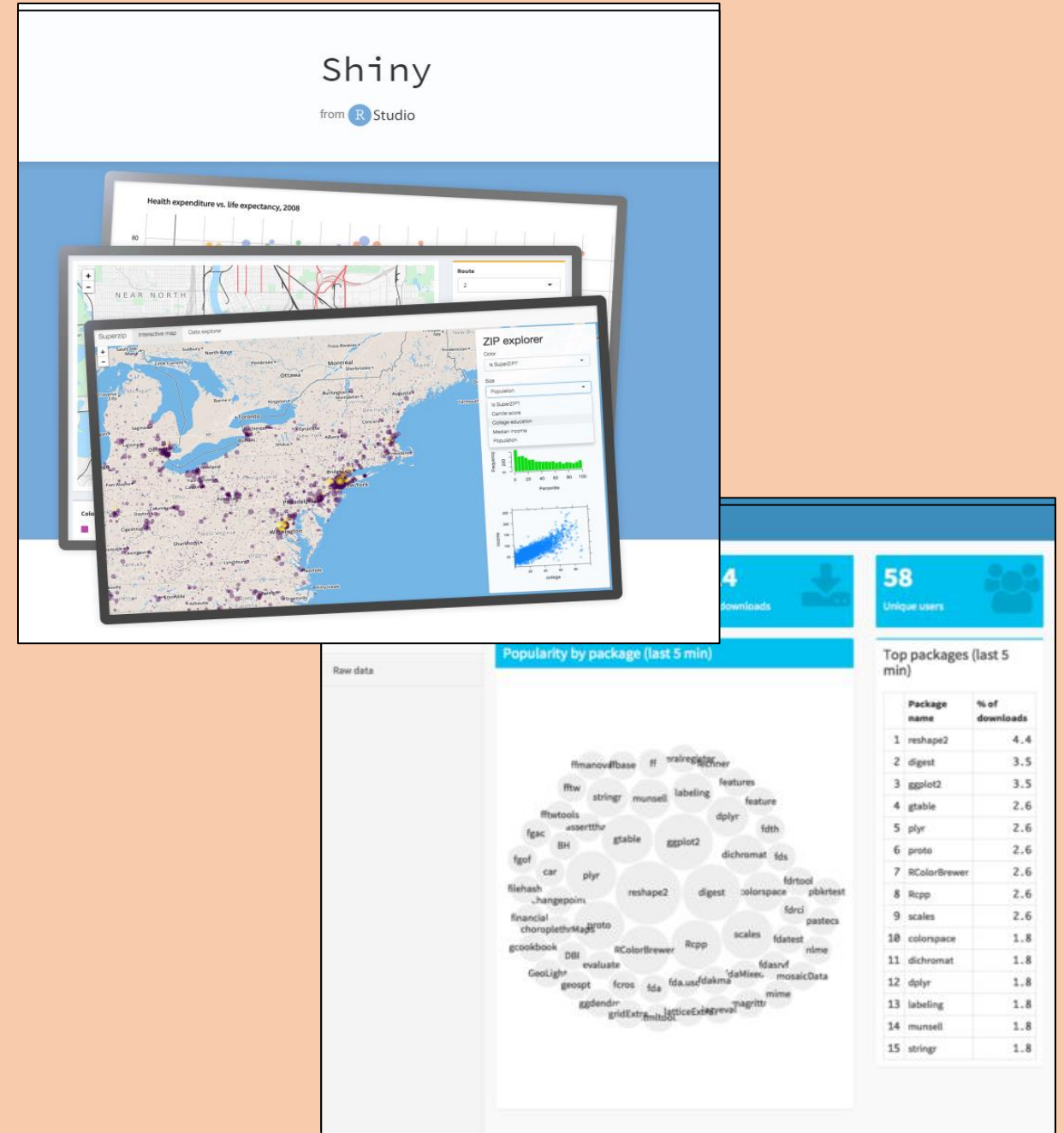
Shiny

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

Shiny combines the computational power of R with the interactivity of the modern web.

You can host standalone apps on a webpage or embed them in R Markdown documents or build dashboards.

You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions.



R Shiny extensions

Additionally, we will cover and use other r-packages that improve the interface and functionalities of Shiny Apps.

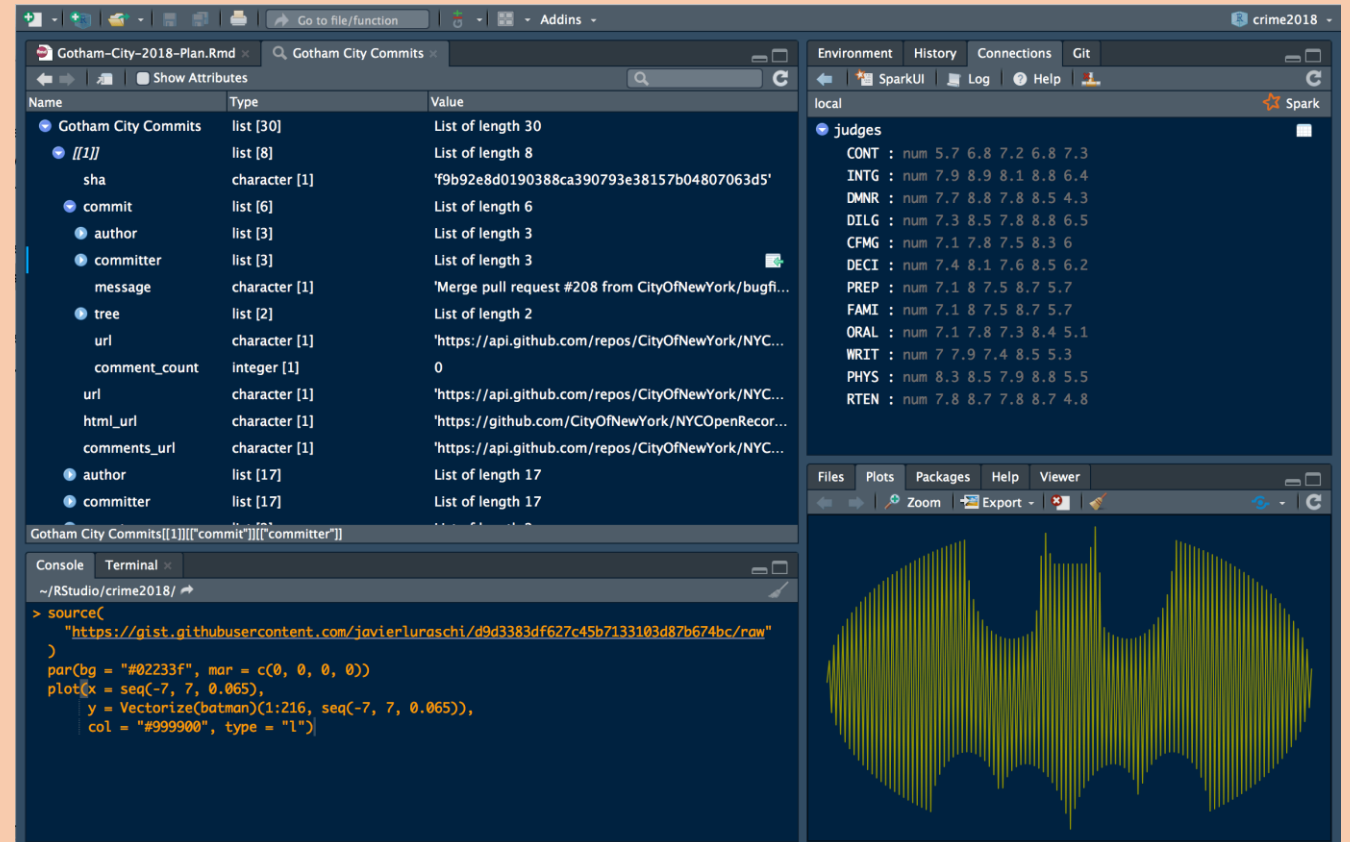
The ones that will cover as follow:

- **shinyjs**: lets you perform common useful JavaScript operations in Shiny apps. [Link](#)
- **shinyjsky**: lets you perform common useful JavaScript operations in Shiny apps. [Link](#)
- **shinyBS**: package that adds several additional Twitter Bootstrap components to shiny . [Link](#)
- **bsplus**: provide access to some useful Bootstrap components for rmarkdown html-documents and shiny apps. [Link](#)

RStudio

RStudio is an integrated development environment (IDE) for R. It includes a **console**, syntax-highlighting **editor** that supports direct code execution, as well as **tools** for plotting, **history**, **debugging** and **workspace management**.

RStudio is available in **open source** and **commercial** editions and runs on the **desktop** (Windows, Mac, and Linux) or in a **browser** connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, RedHat/CentOS, and SUSE Linux).



Desktop

Run RStudio on your desktop

[RStudio Desktop >](#)



Server

Centralize access and computation

[RStudio Server >](#)

Structure

Shiny apps are contained in a single script called **app.R**. The script app.R lives in a directory (for example, newdir/) and the app can be run with **runApp("newdir")**.

app.R has three components:

a user interface object

The user interface (ui) object controls the layout and appearance of your app.

a server function

The server function contains the instructions that your computer needs to build your app

a call to the shinyApp function

Finally the shinyApp function creates Shiny app objects from an explicit UI/server pair.

```
library(shiny)

# Define UI ----
ui <- fluidPage(

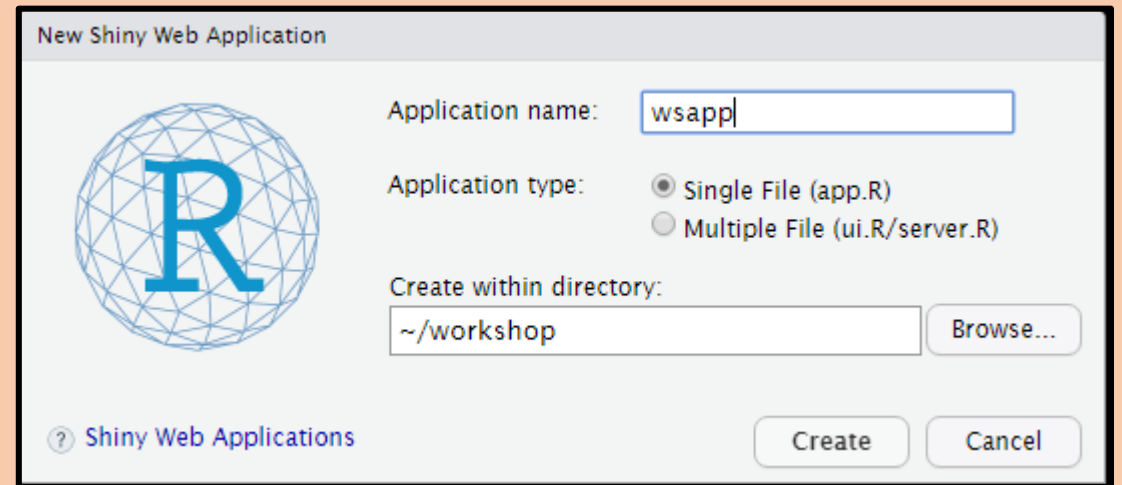
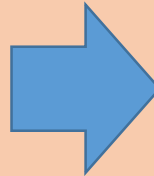
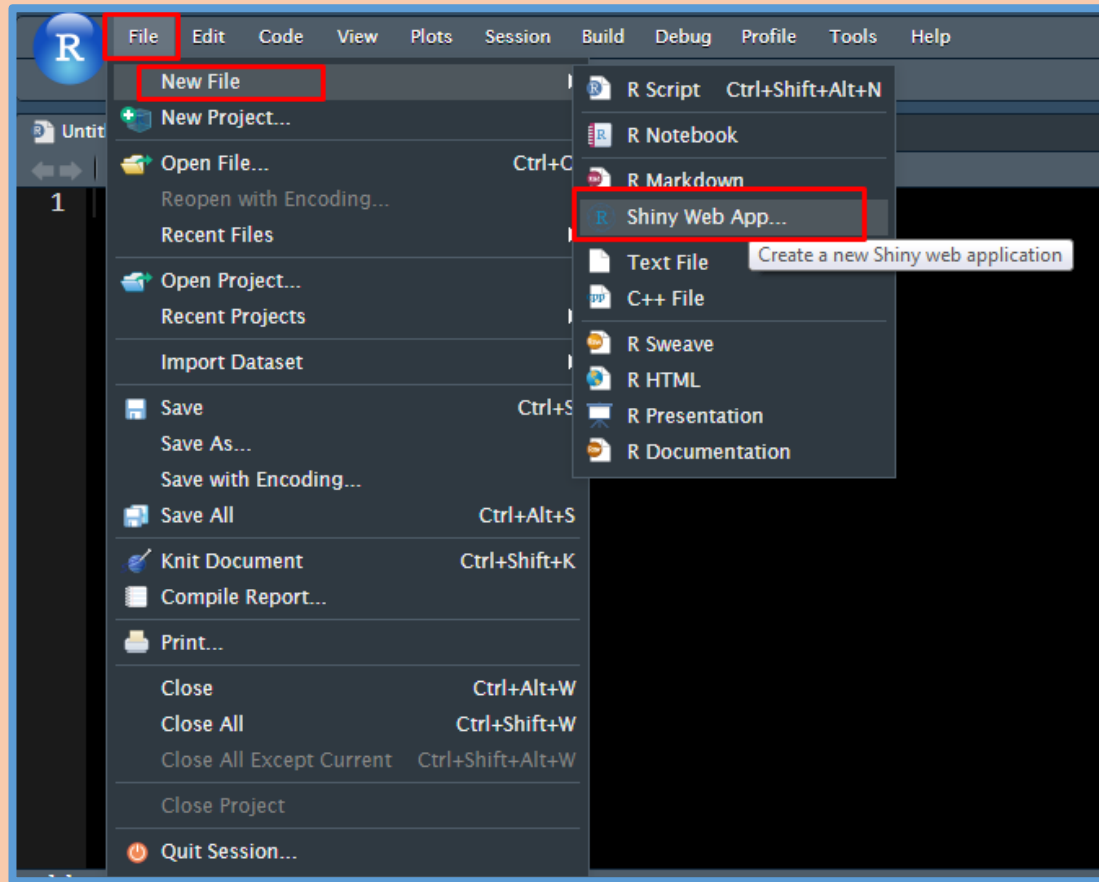
)

# Define server logic ----
server <- function(input, output) {

}

# Run the app ----
shinyApp(ui = ui, server = server)
```

Setup of shiny app



Setup of shiny app

Requirements

- R and Shiny package must installed in your computer or server.
- RStudio for the deployment of the applications.

Steps

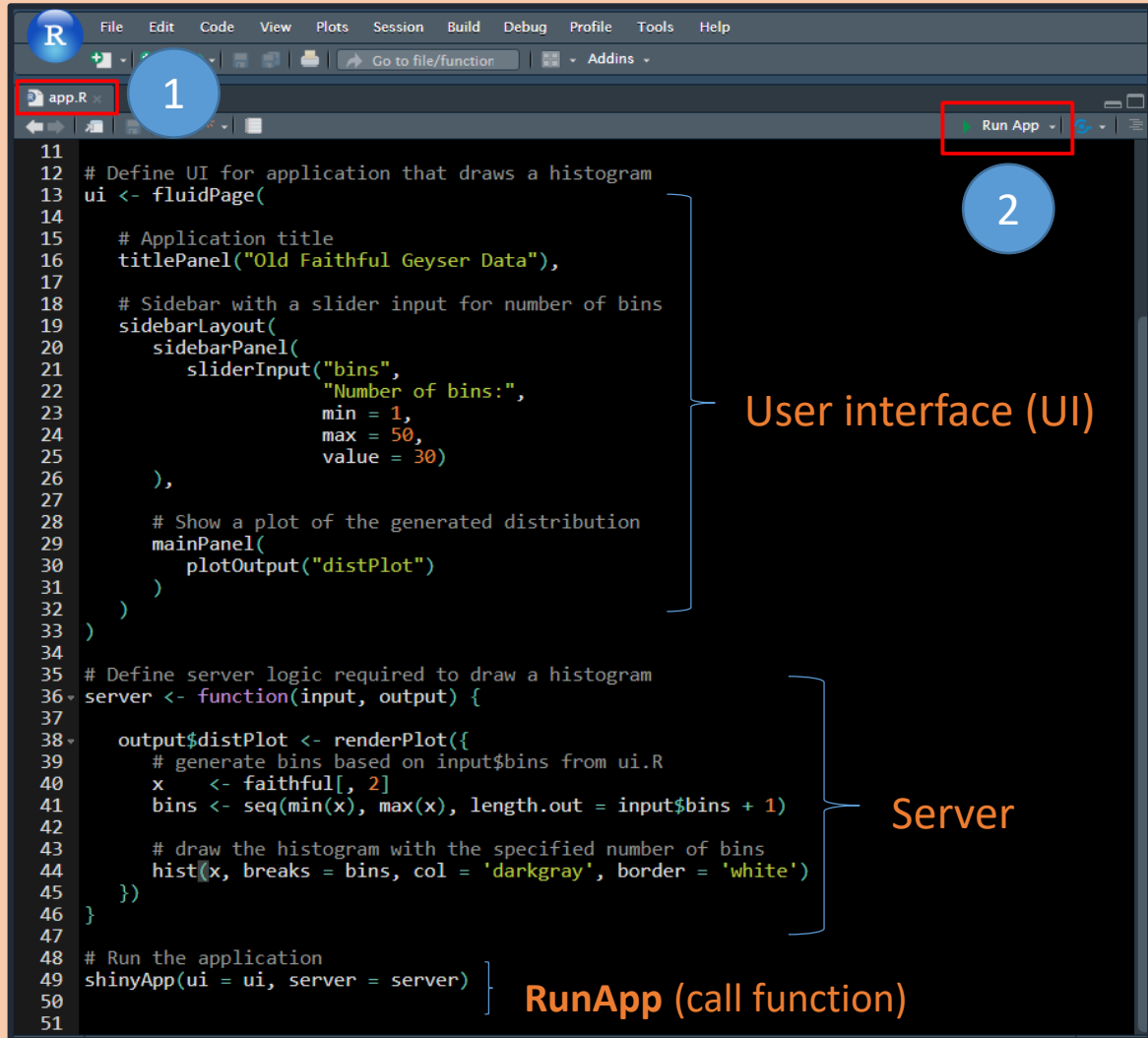
Method #1: One-file App

We can write all the code (ui and server) in a single file **app.R** file. Then, call **runApp(ui,server)** to launch the application in the browser.

Method #2 : Two-files App

Alternatively, Shiny applications have two main files: **ui.R** and **server.R**. Both are for the user interface and the server functionalities respectively. It's highly adopted when the code increase in both sites.

Method #1: One-file App



```
11 # Define UI for application that draws a histogram
12 ui <- fluidPage(
13   # Application title
14   titlePanel("Old Faithful Geyser Data"),
15   # Sidebar with a slider input for number of bins
16   sidebarLayout(
17     sidebarPanel(
18       sliderInput("bins",
19         "Number of bins:",
20         min = 1,
21         max = 50,
22         value = 30)
23     ),
24     # Show a plot of the generated distribution
25     mainPanel(
26       plotOutput("distPlot")
27     )
28   )
29 )
30
31 # Define server logic required to draw a histogram
32 server <- function(input, output) {
33   output$distPlot <- renderPlot({
34     # generate bins based on input$bins from ui.R
35     x <- faithful[, 2]
36     bins <- seq(min(x), max(x), length.out = input$bins + 1)
37
38     # draw the histogram with the specified number of bins
39     hist(x, breaks = bins, col = 'darkgray', border = 'white')
40   })
41 }
42
43 # Run the application
44 shinyApp(ui = ui, server = server)
```

1

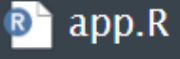
2

User interface (UI)

Server

RunApp (call function)

1

By default, RStudio create  app.R tab containing the ui and server code.

2

In the right side, exist  Run App the button to launch the shiny application.

Method #2 : Two-files App

```
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

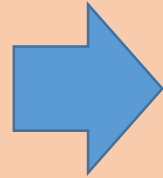
    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
)

# Define server logic required to draw a histogram
server <- function(input, output) {

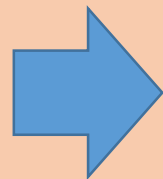
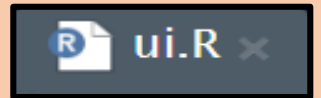
  output$distPlot <- renderPlot({
    # generate bins based on input$bins from ui.R
    x <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
}

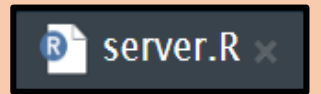
shinyApp(ui = ui, server = server)
```



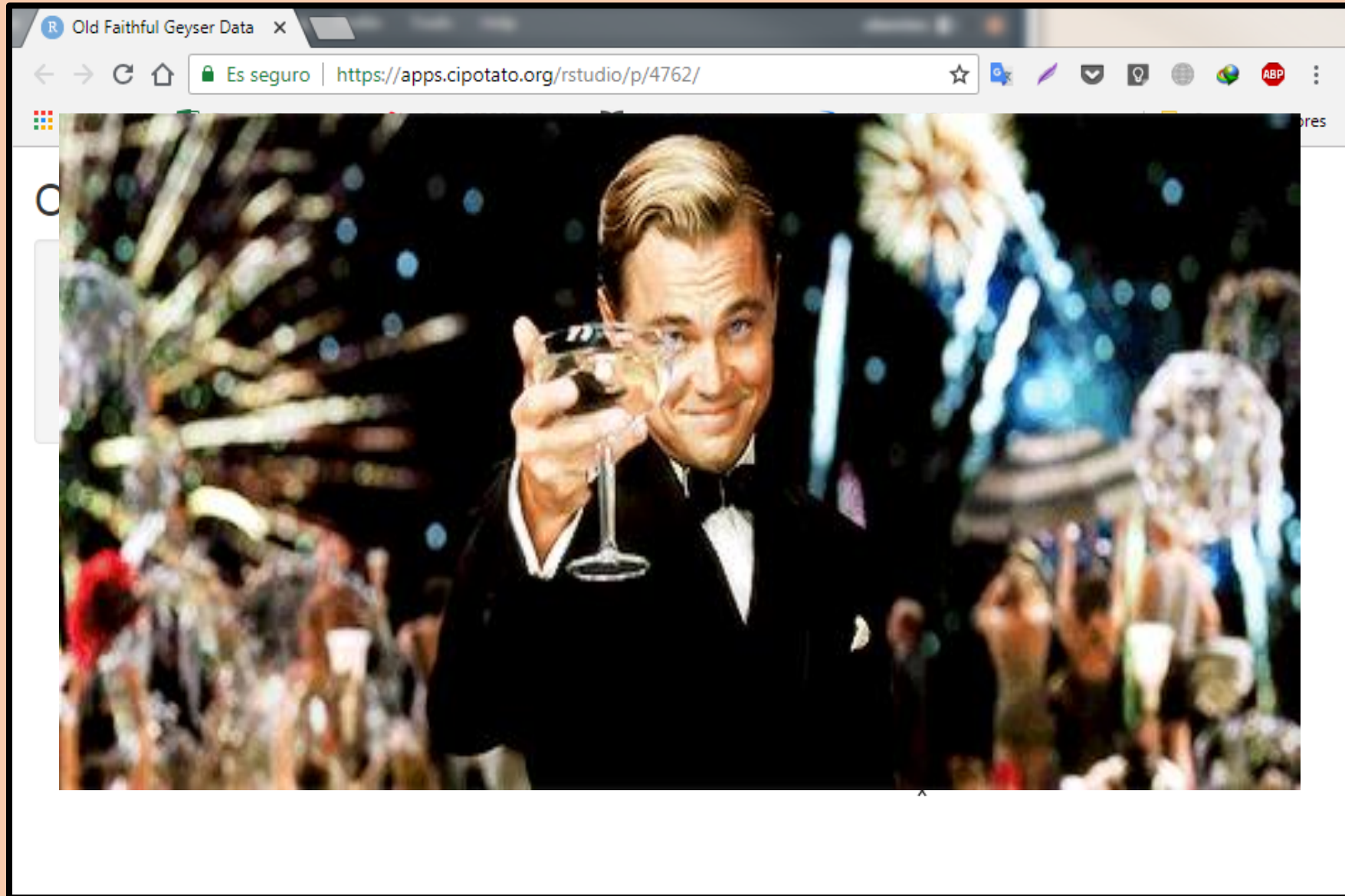
```
1 # ui.R
2 # Define UI for application that draws a histogram
3 ui <- fluidPage(
4   # Application title
5   titlePanel("Old Faithful Geyser Data"),
6   # Sidebar with a slider input for number of bins
7   sidebarLayout(
8     sidebarPanel(
9       sliderInput("bins",
10                  "Number of bins:",
11                  min = 1,
12                  max = 50,
13                  value = 30)
14     ),
15     # Show a plot of the generated distribution
16     mainPanel(
17       plotOutput("distPlot")
18     )
19   )
20 )
```



```
1 # server.R
2
3 # Define server logic required to draw a histogram
4 server <- function(input, output) {
5
6   output$distPlot <- renderPlot({
7     # generate bins based on input$bins from ui.R
8     x <- faithful[, 2]
9     bins <- seq(min(x), max(x), length.out = input$bins + 1)
10
11     # draw the histogram with the specified number of bins
12     hist(x, breaks = bins, col = 'darkgray', border = 'white')
13   })
14 }
```



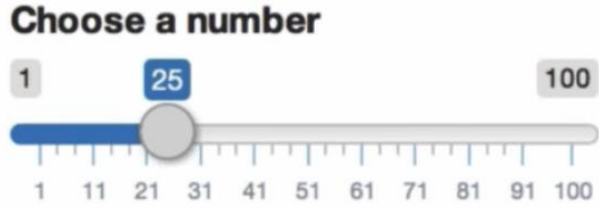
Launch an app



Input elements

Input syntax

Syntax



Choose a number

1 25 100

1 11 21 31 41 51 61 71 81 91 100

```
sliderInput(inputId = "num", label = "Choose a number", ...)
```

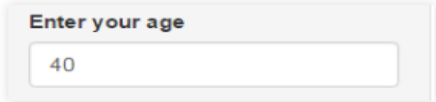
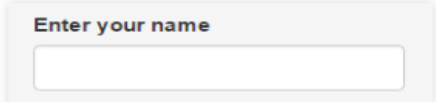
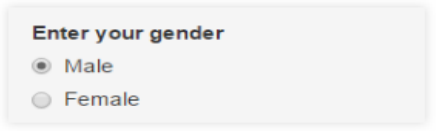


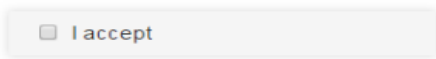
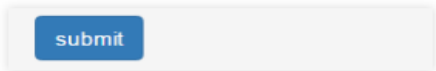
input name
(for internal use)

Notice:
Id not ID

label to
display

Input elements

Shiny Inputs

Type	Function	Arguments	Example
Numeric input	numericInput	<code>inputId, label, value, min, max, step</code>	
Text input	textInput	<code>inputId, label, value, width</code>	
Options list	radioButtons	<code>inputId, label, choices, selected, inline, width</code>	
Drop-down list	selectInput	<code>inputId, label, choices, selected, multiple, selectize, width, size</code>	
Drop-down list	selectizeInput	<code>... + options</code>	
Numeric input (minimum, maximum)	sliderInput	<code>inputId, label, min, max, value, step, animate</code>	
True/False	checkboxInput	<code>inputId, label, value, width</code>	
Button	actionButton	<code>inputId, label, icon, width</code>	

Output elements

Output syntax

server.R

```
output$selected_var <- renderText({  
  "You have selected this"  
})
```

Output ID

Shiny Render
Output

ui.R


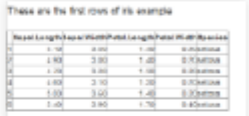
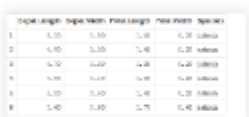

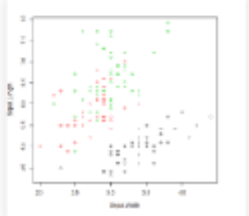
```
mainPanel(  
  textOutput("selected_var")  
)
```

Shiny Output

Output ID

Output elements

Shiny Outputs

Type	Function	Arguments	Example
R-console like text	verbatimTextOutput	outputId	
HTML interpreted text	htmlOutput	outputId, inline	
“Regular” table	tableOutput	outputId	
Dynamic table	dataTableOutput	outputId	
Plots	plotOutput imageOutput	outputId, width, height, click, ...	

Exercise 1

Part II

Reactivity

renderUI

Create a shiny Input (widget) from the server side.

There are cases when the inputs need to be rendered using data that is previously processed in the server side.

```
ui <- fluidPage(  
  uiOutput("moreControls")  
)  
  
server <- function(input, output) {  
  output$moreControls <- renderUI({  
    tagList(  
      sliderInput("n", "N", 1, 1000, 500),  
      textInput("label", "Label")  
    )  
  })  
}  
shinyApp(ui, server)
```

Conditional Panel

Creates a panel that is visible or not, depending on input values.

What do you want to show?

☒ Table

☐ Summary

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.10	3.50	1.40	0.20	setosa
4.90	3.00	1.40	0.20	setosa
4.70	3.20	1.30	0.20	setosa
4.60	3.10	1.50	0.20	setosa
5.00	3.60	1.40	0.20	setosa
5.40	3.90	1.70	0.40	setosa

What do you want to show?

☐ Table

☒ Summary

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	versicol
Median :5.800	Median :3.000	Median :4.350	Median :1.300	virginica
Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.199	
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	
Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500	

```
conditionalPanel(condition =  
  "input.idCbox == 'Table'",  
  ...  
)  
conditionalPanel(condition =  
  "input.idCbox == 'Summary'",  
  ...  
)
```

Reactive Expressions

Shiny comes with a **reactive programming** library that you will use to structure your application logic. By using this library, changing input values will naturally cause the right parts of your R code to be reexecuted, which will in turn cause any changed outputs to be updated.

Reactivity

Caption:

Choose a dataset:

rock

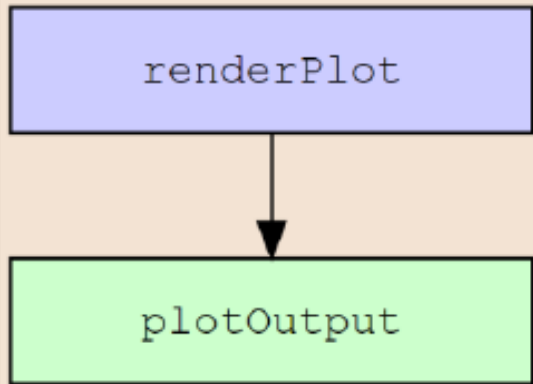
Number of observations to view:

Data Summary

area	peri	shape	perm
Min. : 1016	Min. : 308.6	Min. : 0.09033	Min. : 6.30
1st Qu.: 5305	1st Qu.: 1414.9	1st Qu.: 0.16226	1st Qu.: 76.45
Median : 7487	Median : 2536.2	Median : 0.19886	Median : 130.50
Mean : 7188	Mean : 2682.2	Mean : 0.21811	Mean : 415.45
3rd Qu.: 8870	3rd Qu.: 3989.5	3rd Qu.: 0.26267	3rd Qu.: 777.50
Max. : 12212	Max. : 4864.2	Max. : 0.46413	Max. : 1300.00

	area	peri	shape	perm
1	4990	2791.90	0.09	6.30
2	7002	3892.60	0.15	6.30

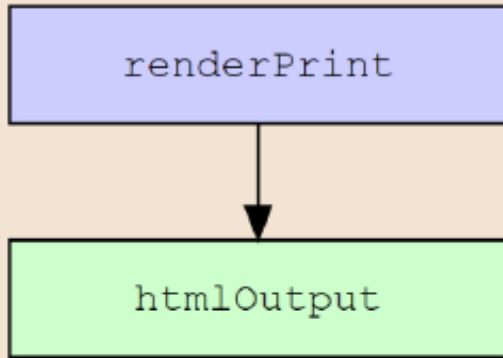
plot Output



```
ui <- fluidPage(  
  plotOutput("result")  
)  
  
server <- function(input, output) {  
  
  output$result<-renderPlot({  
    plot(Sepal.Length ~ Sepal.Width,  
        col = Species, data = iris)  
  }, width = 500, height = 500)  
  
}  
  
shinyApp(ui = ui, server = server)
```

How would the figure look like if width and height are not specified?

html Output



```
library(xtable)

ui <- fluidPage(
  htmlOutput("result")
)

server <- function(input, output) {
  output$result<-renderPrint({
    print(xtable(head(iris)), type = "html")
  })
}

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

Exercise: Replace htmlOutput by
verbatimTextOutput

DT Library: DataTable

Dynamic table: extensions

- No need to plug into Shiny app
- Wrap the data.frame using **datatable** function from **DT** package to add more options (filter, rownames, download buttons, ...)
- Execute the following code in R-script or R-markdown.

```
library(DT)
DT::datatable(iris, filter="top", rownames=FALSE,
  extensions = 'Buttons',
  options = list(dom = 'Bfrtip',
    buttons = c('copy', 'csv', 'excel', 'pdf', 'print')
  )
)
```

DT Library: DataTable

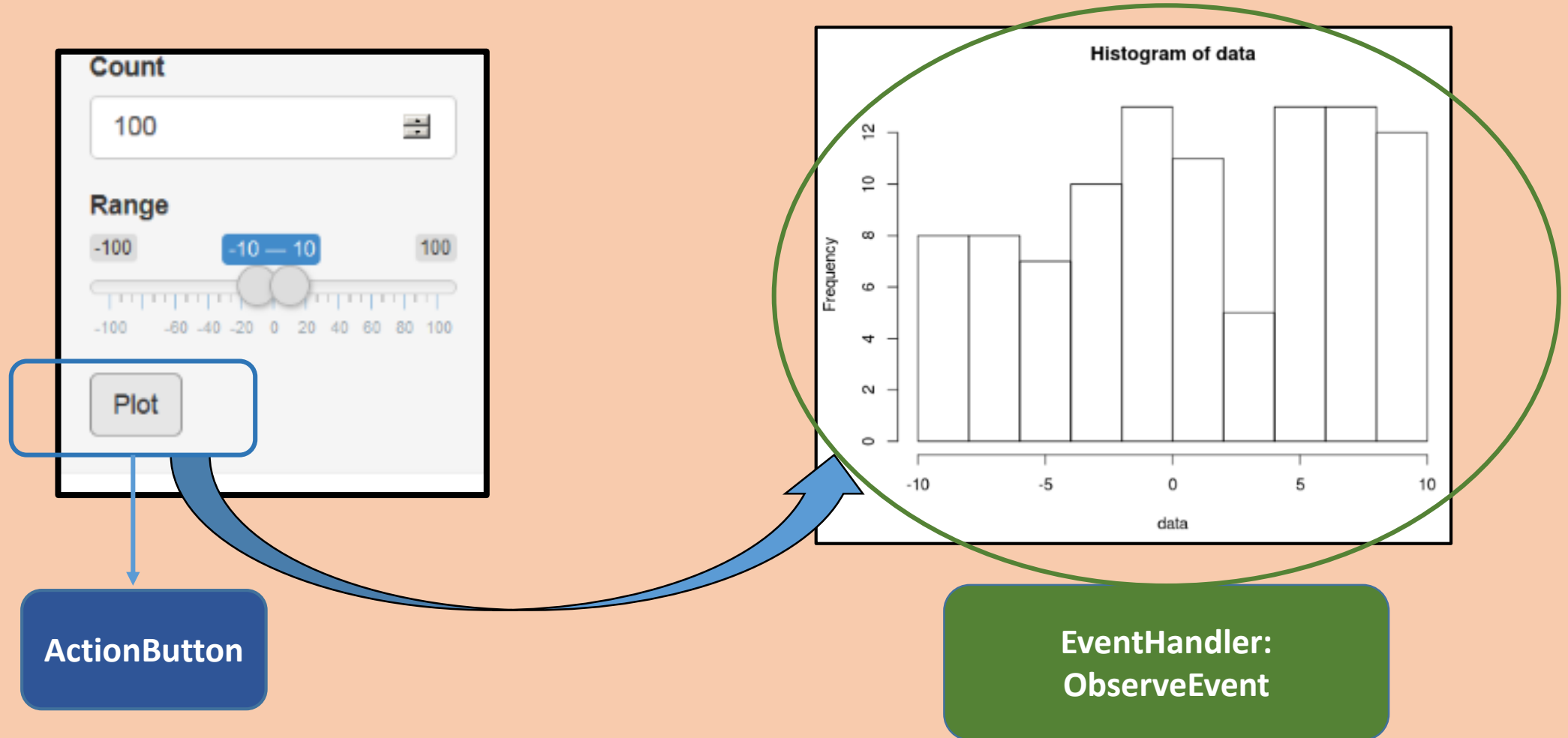
renderDataTable



dataTableOutput

```
ui <- fluidPage(  
  dataTableOutput("result")  
)  
  
server <- function(input, output) {  
  output$result<-renderDataTable({  
    iris  
  })  
}  
  
shinyApp(ui = ui, server = server)
```

ActionButton and ObserveEvent



rhansontable Library: R-Based Spreadsheet table

renderRhansontable



tableOutput

```
library(rhansontable)
library(shiny)

ui<- fluidPage(
  rHandsontableOutput("dtrhand")
)

server<-function(input, output){

  output$dtrhand <- renderRHandsontable({
    DF = iris
    if (!is.null(DF))
      rhansontable(DF)
  })

}

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

Exercise 3

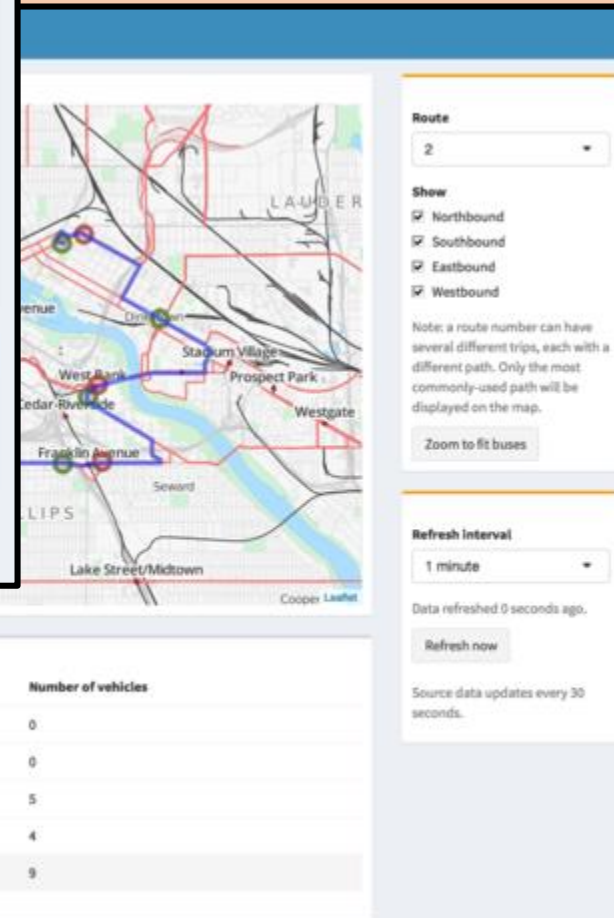
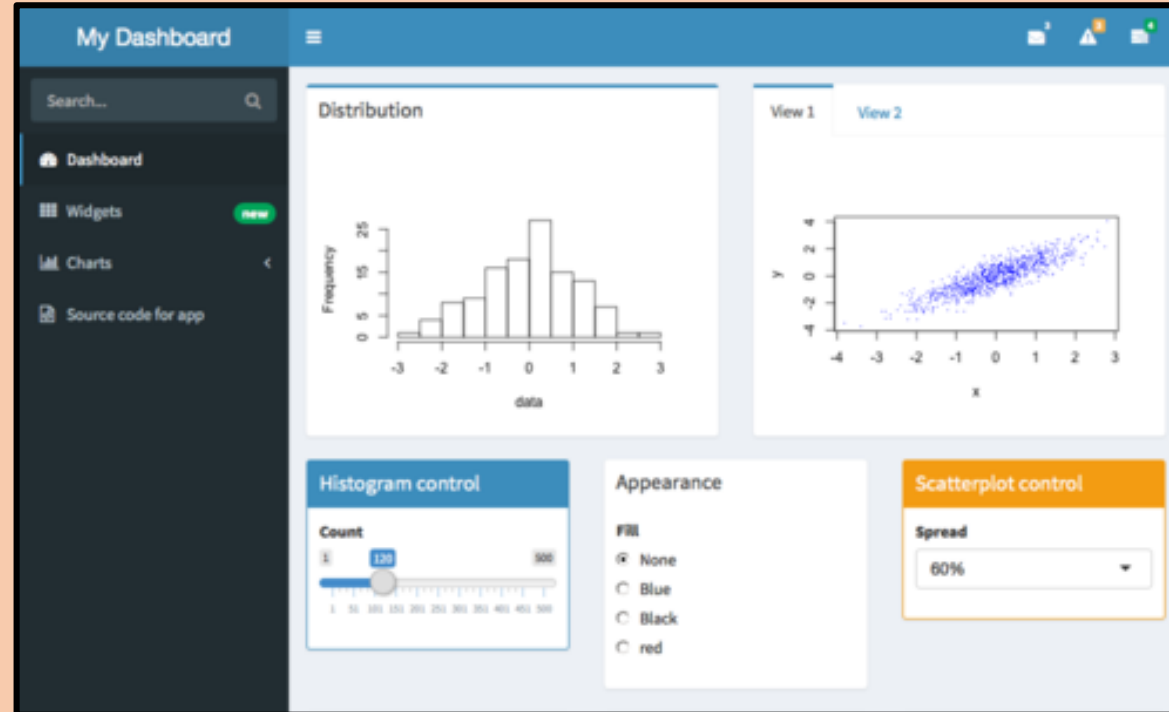
Part III

Shiny Dashboard

Shiny Dashboard

shinydashboard is an R package whose job is to make it easier (as the name suggests) to build dashboards with shiny.

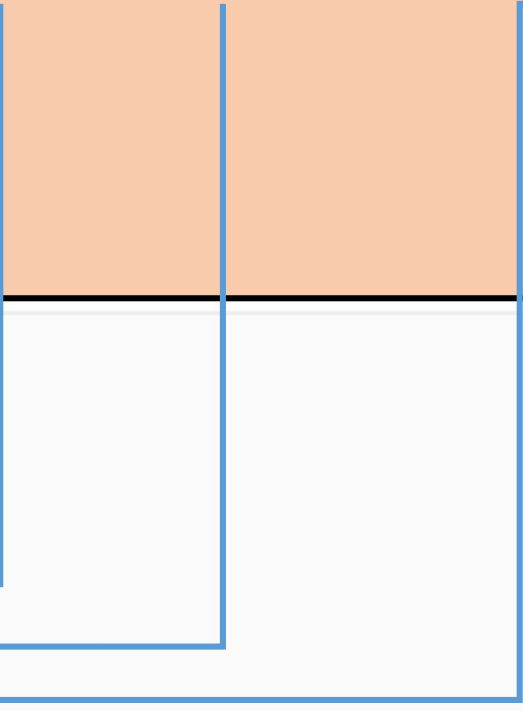
shinydashboard makes it easy to use Shiny to create dashboards like these:



Shiny Dashboard

A dashboard has three parts: a header, a sidebar, and a body. Here's the most minimal possible UI for a dashboard page.

```
## ui.R ##  
library(shinydashboard)  
  
dashboardPage(  
  dashboardHeader(),  
  dashboardSidebar(),  
  dashboardBody()  
)
```



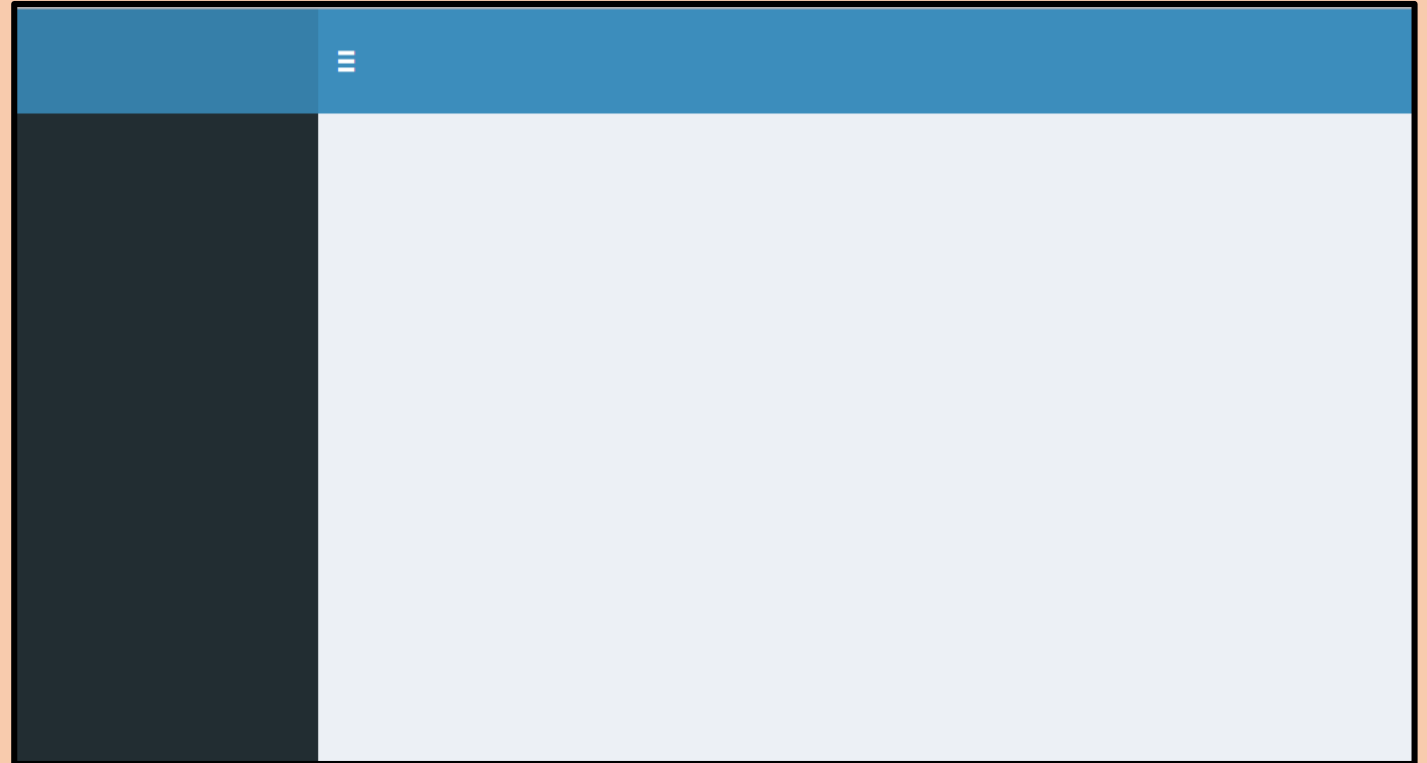
The diagram consists of three vertical blue lines originating from the text 'A dashboard has three parts: a header, a sidebar, and a body.' and ending with arrows pointing to the arguments of the `dashboardPage()` function in the code block. The first line points to `dashboardHeader()`, the second line points to `dashboardSidebar()`, and the third line points to `dashboardBody()`.

Shiny Dashboard

Basics

You can quickly view it at the R console by using the `shinyApp()` function. (You can also use this code as a single-file app).

```
## app.R ##  
library(shiny)  
library(shinydashboard)  
  
ui <- dashboardPage(  
  dashboardHeader(),  
  dashboardSidebar(),  
  dashboardBody()  
)  
  
server <- function(input, output) { }  
  
shinyApp(ui, server)
```

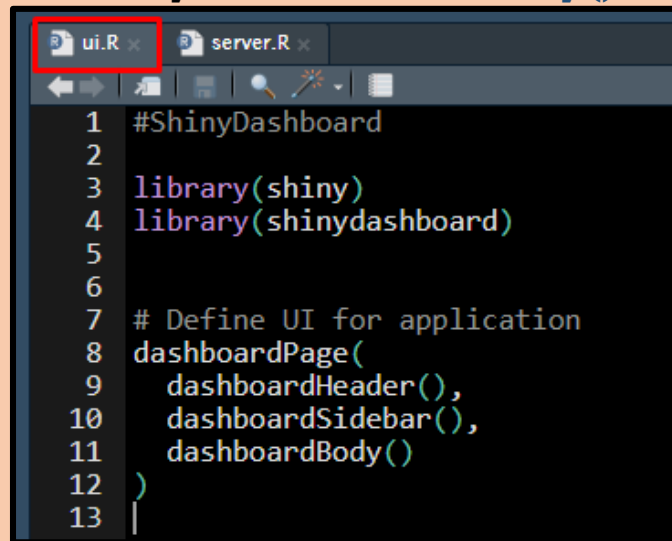


Shiny Dashboard

UI & Server

UI

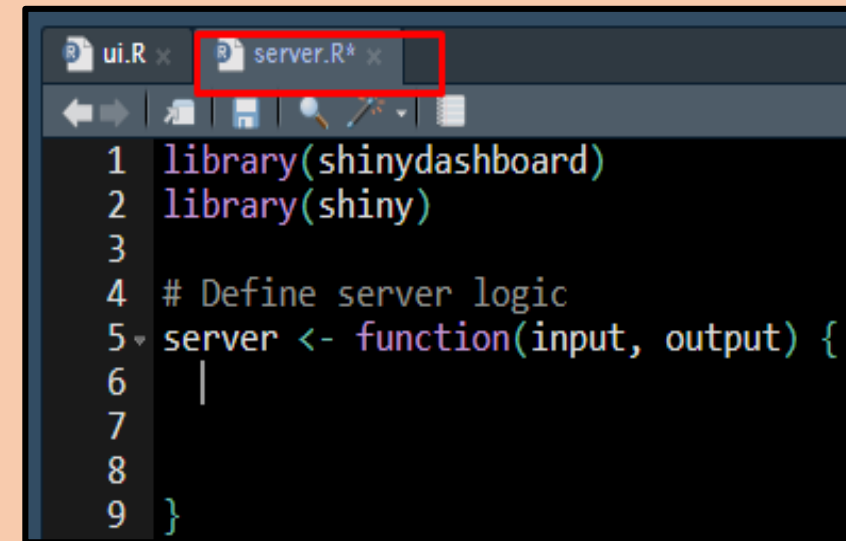
- Define Dashboard Page: **dashboardPage()**
- The dashboard page is divided in 3 elements:
 - Header: **dashboardHeader()**
 - Sidebar: **dashboardSidebar()**
 - Body: **dashboardBody()**



```
1 #ShinyDashboard
2
3 library(shiny)
4 library(shinydashboard)
5
6
7 # Define UI for application
8 dashboardPage(
9   dashboardHeader(),
10  dashboardSidebar(),
11  dashboardBody()
12 )
13 |
```

Server

- Where all computations are performed.
- Render inputs and reactive values are placed here.



```
1 library(shinydashboard)
2 library(shiny)
3
4 # Define server logic
5 server <- function(input, output) {
6   |
7
8
9 }
```

Shiny Dashboard

Header

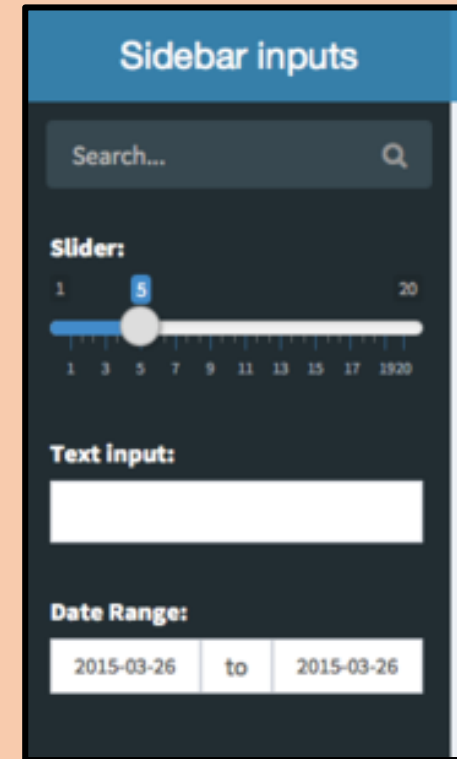
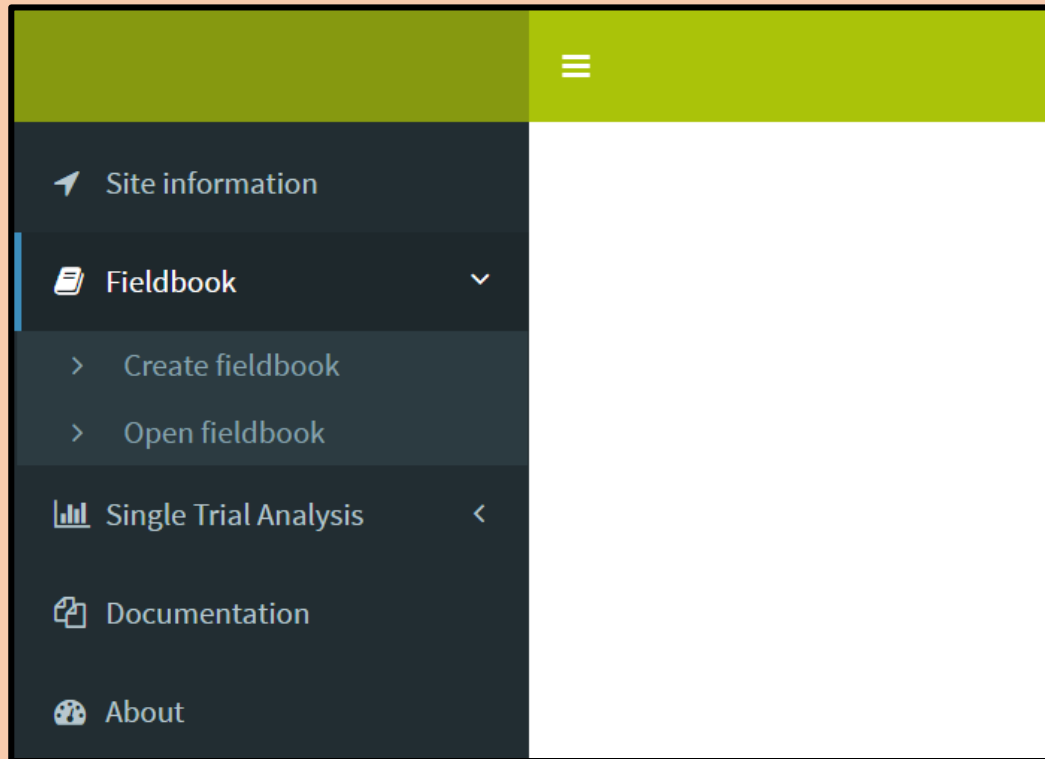
A header can have a title, dropdown menus, login, etc. Here's an example:



Shiny Dashboard

Sidebar

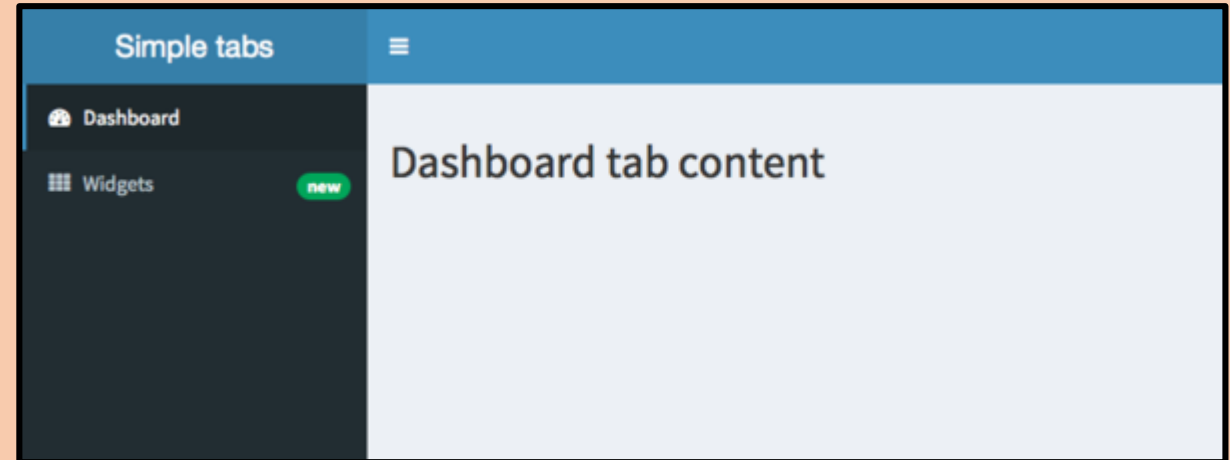
A sidebar is typically used for quick navigation. It can contain menu items that behave like tabs in a `tabPanel`, as well as Shiny inputs, like sliders and text inputs.



Shiny Dashboard

Sidebar menu items and tabs

Links in the sidebar can be used like `tabPanels` from Shiny. That is, when you click on a link, it will display different content in the body of the dashboard. Here is an example of a simple `tabPanel`:



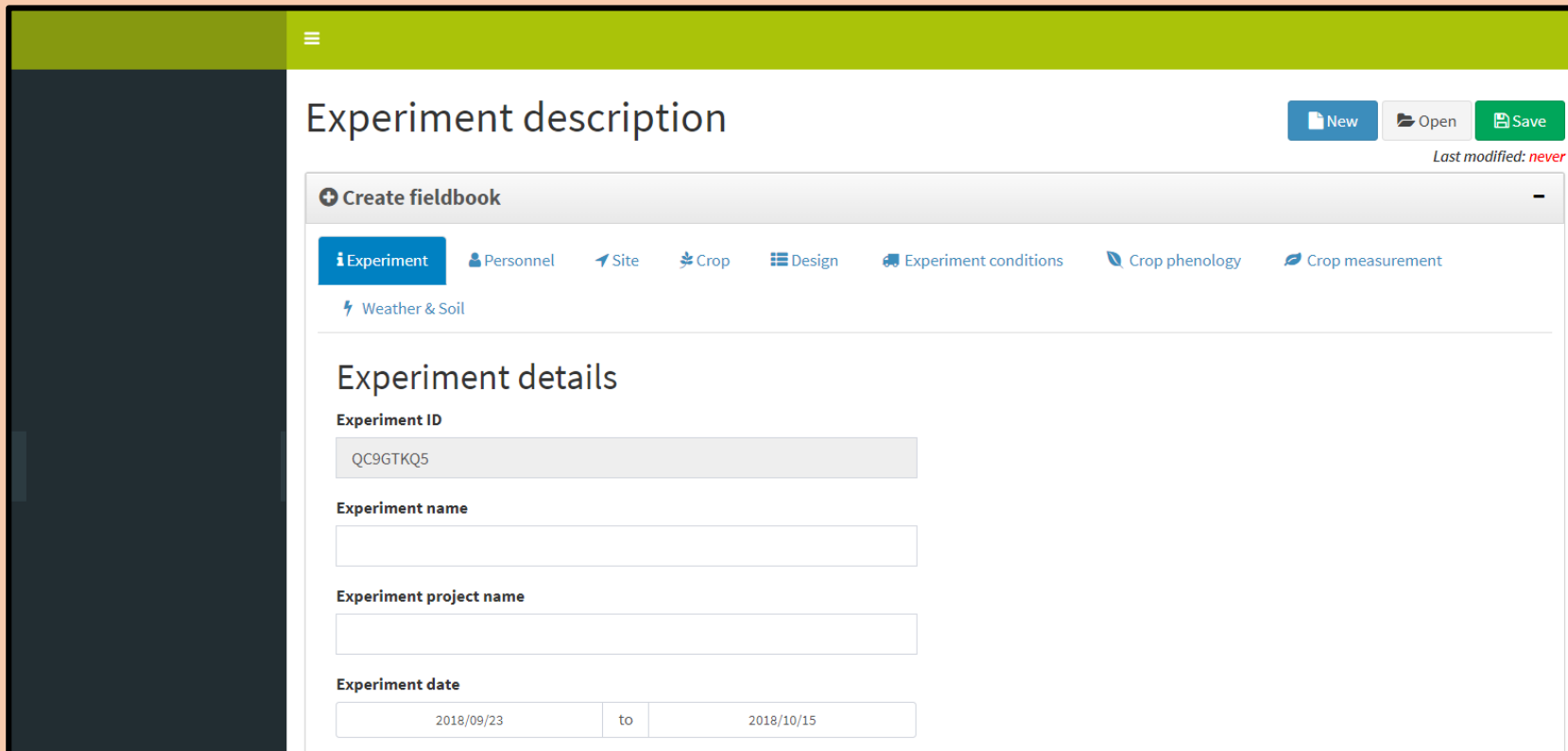
- `sidebarMenu`:
 - contains menu items
- `menuItem`:
 - side bar item into dashboard
- `menuSubItem`:
 - sub items inside menu item

```
1 #ShinyDashboard
2
3 library(shiny)
4 library(shinydashboard)
5
6
7 # Define UI for application
8 dashboardPage(
9   dashboardHeader(title = "Title here"),
10  dashboardSidebar(
11    # SIDE BAR ITEMS GO HERE
12    sidebarMenu(id = "tabs",
13               menuItem(text = "dashboards", icon = icon("dashboard")),
14               menuItem(text = "Charts", icon = icon("bar-chart-o"),
15                      menuSubItem("Sub-item 1", tabName = "subitem1"),
16                      menuSubItem("Sub-item 2", tabName = "subitem2"))
17    )
18  ),
19  dashboardBody(
20    # CONTENT GOES HERE
21  )
22 )
```

Shiny Dashboard

Body

The body of a dashboard page can contain any regular Shiny content. However, if you're creating a dashboard you'll likely want to make something that's more structured. The basic building block of most dashboards is a box. Boxes in turn can contain any content.

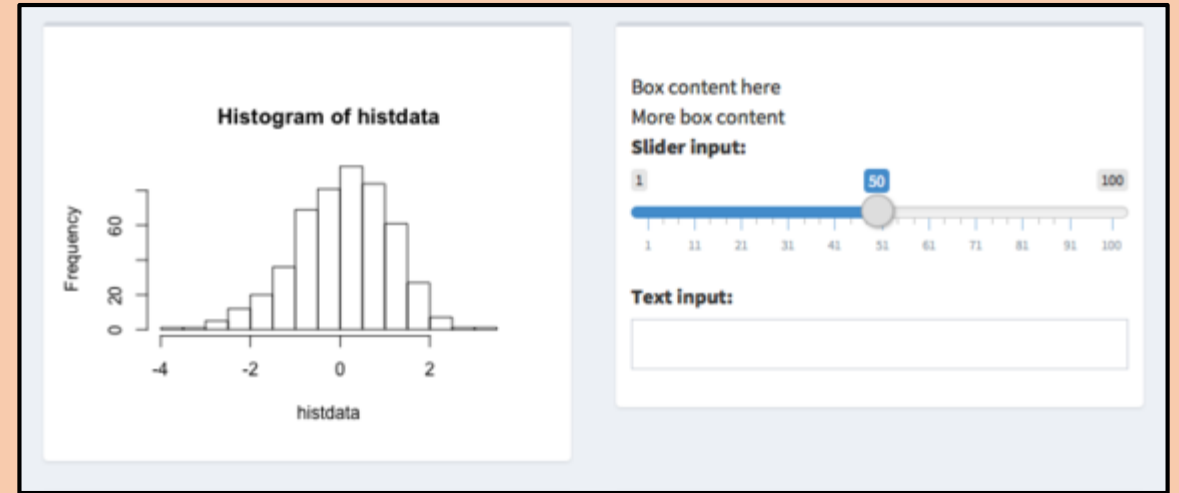


The screenshot shows a Shiny dashboard interface. At the top, there is a green header bar with a hamburger menu icon. Below the header, the main content area is titled "Experiment description". To the right of the title are three buttons: "New" (blue), "Open" (grey), and "Save" (green). Below these buttons, it says "Last modified: never" in red. A tab bar is present with the following tabs: "Create fieldbook" (active), "Experiment", "Personnel", "Site", "Crop", "Design", "Experiment conditions", "Crop phenology", and "Crop measurement". Below the tabs, there is a section titled "Weather & Soil" with a lightning bolt icon. The main content area is titled "Experiment details" and contains several input fields: "Experiment ID" (with the value "QC9GTKQ5"), "Experiment name", "Experiment project name", and "Experiment date" (with a date range from "2018/09/23" to "2018/10/15").

Shiny Dashboard

Boxes

Boxes are the main building blocks of dashboard pages. A basic box can be created with the `box()` function, and the contents of the box can be (most) any Shiny UI content.



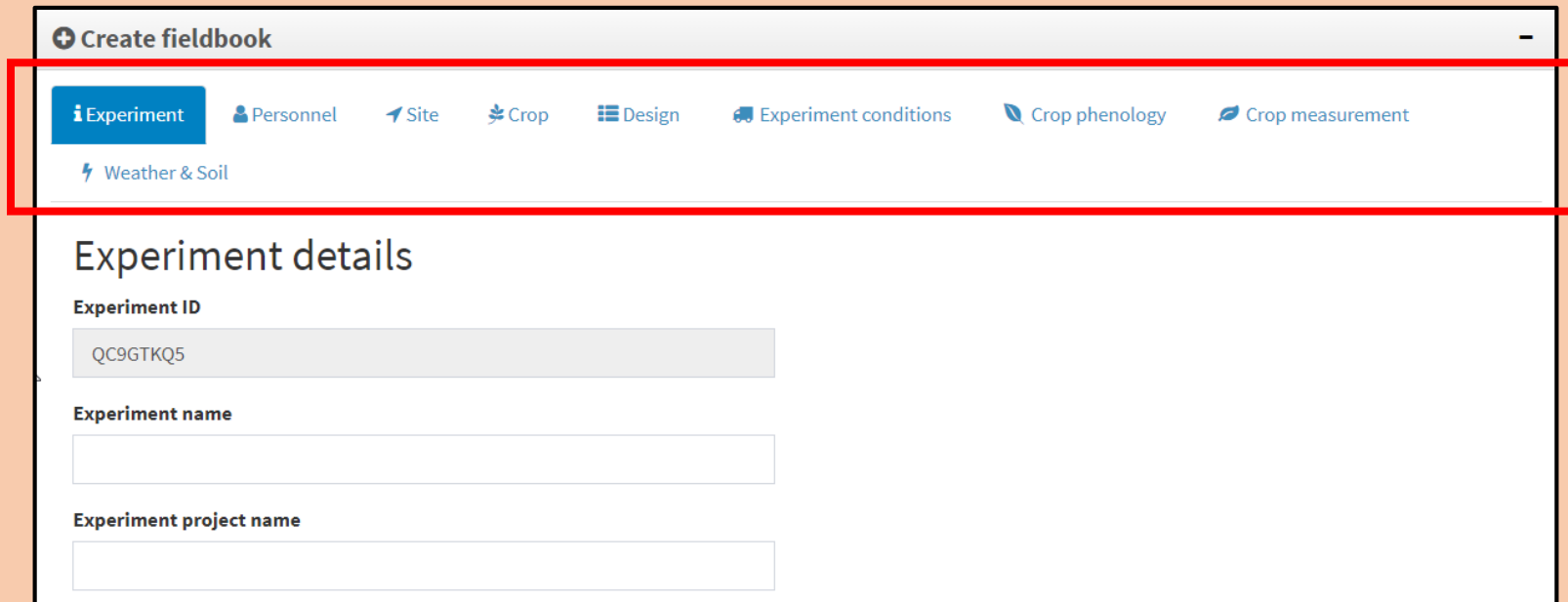
This figure shows a Shiny dashboard interface. At the top, there is a blue header bar. Below it, there is a tab labeled "Single Analysis". The main content area features a green success message box with a white checkmark icon and the text "GREAT! Now you are connected to SweetPotatoBase using BrAPI". Below this message, there are three selection controls: "Select program" with a dropdown menu showing "Ghana", "Select trial" with a dropdown menu showing "2010", and "Select study" with a dropdown menu showing "Choose".

Shiny Dashboard

tabBox

If you want a box to have tabs for displaying different sets of content, you can use a `tabBox`.

A `tabBox` also has similarities to a regular box from `shinydashboard`, in that you can control the height, width, and title. You can also choose which side the tabs appear on, with the `side` argument. Note that if `side="right"`, the tabs will be displayed in reverse order.



Create fieldbook

Experiment Personnel Site Crop Design Experiment conditions Crop phenology Crop measurement

Weather & Soil

Experiment details

Experiment ID

QC9GTKQ5

Experiment name

Experiment project name

Shiny Dashboard

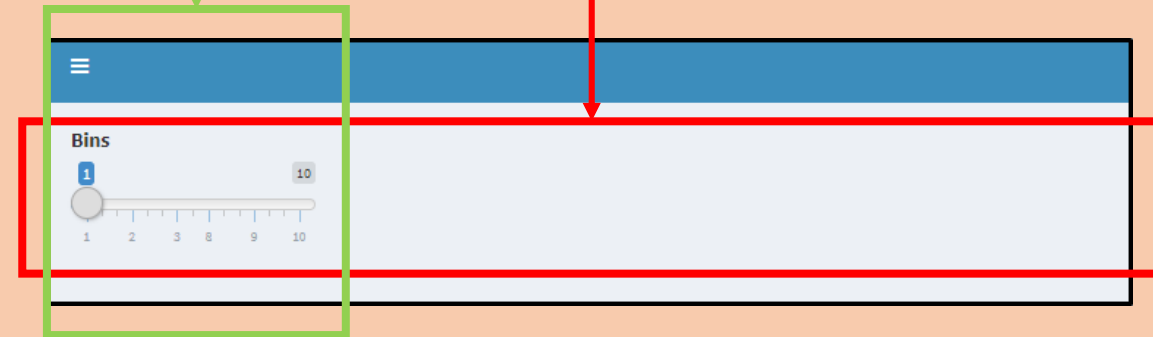
Layouts

The body can be treated as a region divided in to 12 columns of equal width, and any number of rows, of variable height. When you place a box (or other item) in the grid, you can specify how many of the 12 columns you want it to occupy.

Broadly speaking, there are two ways of laying out boxes: with a row-based layout, or with a column-based layout.

- **fluidRow**: row generator. It is the container for **columns**.
- **column**: column generator. It is the container for **widgets**.

```
dashboardBody(  
  # OUPUT GOES HERE  
  fluidRow( #GENERATE ROWS  
  
    column(4, #GENERATE COLUMNS  
      shiny::sliderInput(inputId = "bins", label = "Bins",  
        min = 1, max = 10, value=0.05)  
    )#end column  
  )#end fluidRow
```



Exercise 4

Part IV
**How to customize
appearance**

Shiny Dashboard

Skins

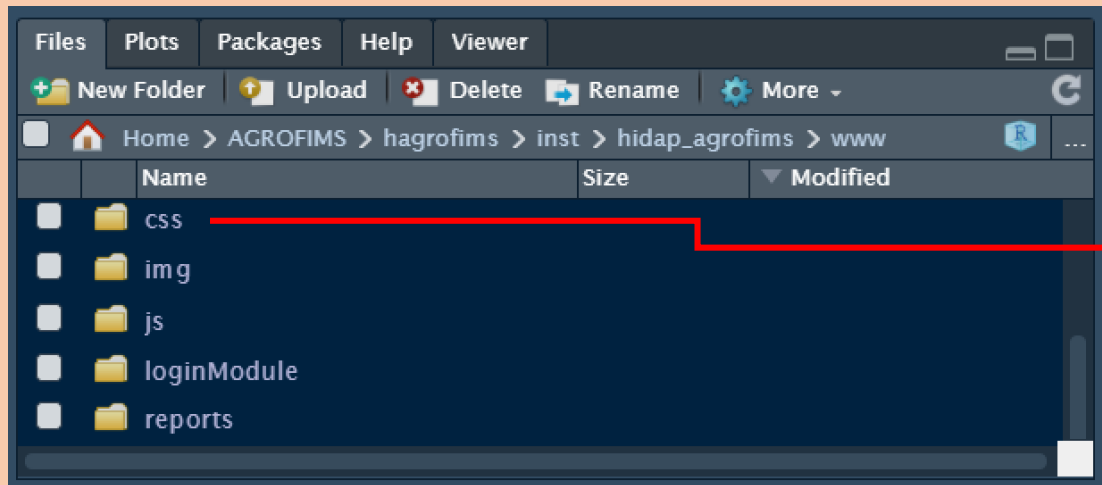
There are a number of color themes, or skins. The default is blue, but there are also black, purple, green, red, and yellow. You can choose which theme to use with `dashboardPage(skin = "blue")`, `dashboardPage(skin = "black")`, and so on.



Shiny Dashboard

CSS

You can add custom CSS to your app by creating a **www/** subdirectory to your app and adding a CSS file there. Suppose, for example, you want to change the title font of your dashboard to the same font as the rest of the dashboard, so that it looks like this:





```
/* Skin */  
.skin-blue .main-header .navbar .dropdown-menu li a {  
  color: #3c8dbc;  
}
```




Shiny Dashboard

More examples...

 Profile

 Authentication

 Log Out

Log in to HIDAP AgroFIMS

Email

none

Password

....


☒ Remember me

Close

Log in

Forgot your password? | Sign up

© 2018 RIU Team. | All Rights Reserved | Terms Of Use



Saved successfully

Harvest details

Start date

2018/09/25

End date

2018/09/25

Harvest cut height

Unit

Select one...

Harvest method

Select one...

Crop component harvested

Select one...

Space between rows harvested

Unit

Select one...

Total area harvested

Unit

Select one...

Implement

Technique

Select one...

Harvest implement

Select one...

Traction

Select one...

Amount harvested

Unit

Select one...

primary

success

info

warning


danger

New

Open

Save

Last modified: never



GREAT!

Now you are connected to SweetPotatoBase using BrAPI

Part V

Sharing apps

Put Shiny Web Apps Online

RStudio lets you put shiny web applications and interactive documents online in the way that works best for you.

Category	Description	RStudio Connect	Shiny Server Pro	Shiny Server Open Source	Shinyapps.io
Overview	Commercial License (not AGPL)	●	●		●
	RStudio Support	●	●		●
	Deploy Shiny applications to the Web	●	●	●	●
	Push-button publishing from RStudio IDE	●			●
	Deploy and access shiny apps, dashboards, R Markdown reports, static plots, and APIs in one place	●			

shinyapps.io

You don't need to own a server or know how to configure a firewall to deploy and manage your applications in the cloud. No hardware, installation.

The image is a composite showing the workflow for deploying a Shiny application to shinyapps.io. It includes the shinyapps.io website, a login modal, the RStudio interface, and the shinyapps.io dashboard.

shinyapps.io Website: The website header includes "shinyapps.io by RStudio" and navigation links: Home, Features, Pricing, Support, and Log In. The main content area says "Share your Shiny Applications Online" and "Deploy your Shiny applications on the Web in minutes". A "Sign Up" button is visible.

Login Modal: A modal window for logging in. It includes fields for Email and Password, a "Log in" button, and links for "Log in with Google" and "Log in with GitHub". There are also links for "Forgot your password?" and "Sign up". The footer of the modal says "© 2017 RStudio Inc. | All Rights Reserved | Terms Of Use".

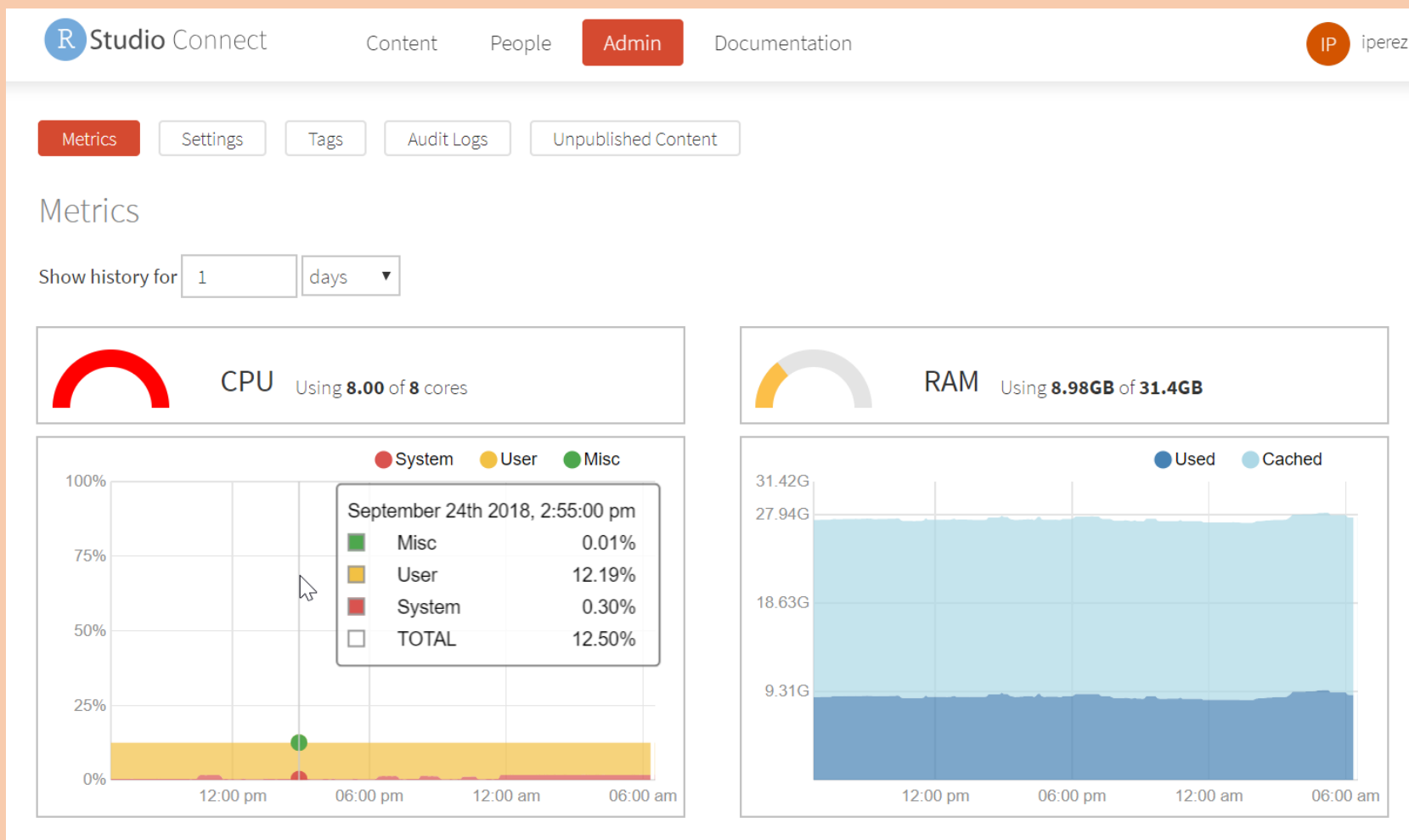
RStudio Interface: The RStudio interface shows the "Run App" button and the "Publish Application..." button. A red arrow points from the "Publish Application..." button to the dashboard.

shinyapps.io Dashboard: The dashboard shows the user's account information (Account: testivan) and a list of applications. The "WHAT'S NEW?" section shows "4 APPLICATIONS ONLINE". The "RECENT APPLICATIONS" section shows a table of applications:

Id	Name	Status
332219	test2	Sleeping
333835	agrofims	Undeployed
333076	agrofims1	Sleeping
331306	test1	Sleeping

RStudio Connect

RStudio Connect is a new publishing platform for the work your teams create in R. Share Shiny applications, R Markdown reports, Plumber APIs, dashboards, plots, and more in one convenient place.



Thank you