MZES SSDL - Shiny Apps: Development and Deployment

Konstantin Gavras

MZES, University of Mannheim

15 October 2019

Shiny

Shiny from RStudio

- Shiny is a package by RStudio to build interactive web pages...
  - without having any knowledge of web development (HTML/CSS/JavaScript)
- Shiny Apps interact with R
  - Allows for calculations, display of R objects, presentation of results ...
- Examples: Democracy and MeTwo

Shiny App Components

1. Front end
   - the web page actually shown to the user
   - the HTML page written by Shiny
   - includes layout, appearance, design features
   - in Shiny terminology: `ui` (user interface)
2. Back end
   - code running the app, including all functions, data import, etc.
   - involves the logic of the app
   - responsible for creating objects on the front end
   - in Shiny terminology: `server`

Setting up a Shiny App

Shiny Apps can be set up in two different ways:

1. Single file App
   - `ui` and `server` are stored in one script
   - used when developing very simple Shiny Apps
   - name of the file has to be `app.R`!!!
2. Two file App
   - `ui` and `server` are stored in separate scripts
   - clear separation between front end and back end
   - highly preferable when developing more advanced Shiny Apps
   - names of the files have to be `ui.R` and `server.R`!!!

$\rightarrow$ We are going to develop Shiny Apps using the Two File method

Developing Shiny Apps - Step by Step

Let's get started!

Workshop materials:
https://github.com/KostaGav/shiny-development-deployment

Features covered in the workshop:

Development:

1. Building a Shiny App from scratch
2. Building the plain UI
3. Getting output objects and control widgets into the UI
4. Implementing the server logic
5. Output/Input Reaction
6. Rendering objects
7. Reactivity

Deployment:

1. Deploy your app using `shinyapps.io`
2. Deploy on your own VM using `Shiny Server`

Building a Shiny App from scratch

```
install.packages("shiny")
library(shiny)
runExample("01_hello")
#To show alternative Apps, please type runExample(NA)
#and choose another example
```

Building a Shiny App from scratch

- Create a new folder with two R scripts:

`ui.R`:

```
library(shiny)
ui <- fluidPage()
```

`server.R`:

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

- Launch the Shiny App by pressing the 'Run App' button in the top right corner

Building the plain UI

Building the plain UI

- When building a Shiny App, one should have, in general, in mind how the app should 'look' like
- Thus, we build the UI first
- In simple Shiny App, the whole UI fits in the `fluidpage`
  - every new object is passed comma-separated
  - text can be passsed to the UI by entering strings
- In order to format text, Shiny uses HTML wrappers:
  - these wrappers are functions taking one object as argument (+ further style options)
  - `h1()`: Top-level header
  - `h2()`: secondary header
  - `strong()`: make text bold
  - `em()`: make text italicized
  - `br()`: add line break
- We can add an official header using `titlePanel()`

Q: Can you see any particular differences between using `h1()` and the `titlePanel()` when using them as title?

Building the plain UI

- Until now, we only have a plain white page
- We need a proper layout to make it appear nicer:
  - `sidebarlayout` is the simplest layout format
  - Input and control widgets on the left side, results and plots on the right hand side

```
ui <- fluidPage(
  titlePanel("Title of my Shiny App"),
  sidebarLayout(
    sidebarPanel("My input goes here"),
    mainPanel("The results go here")
  )
)
```

Adding Input and Control Widgets

- In order to interact with Shiny Apps, we need control widgets
- User can specify inputs, enter text or select specific dates to create a certain results
- all input function have two arguments: `inputId` and `label`
  - `inputId`: name Shiny uses to refer to this input, when retrieving values for the back end
  - has to be unique! (WARNING: if you provide two Ids with the same name, there won't be an error message!)
  - `label`: Text displaying the label of the control widget

Adding Input and Control Widgets

Button

Action

actionButton()

Single checkbox

Choice A

checkboxInput()

Checkbox group

Choice 1
Choice 2
Choice 3

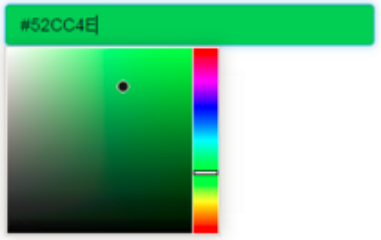
checkboxGroupInput()

Date input

2014-01-01

dateInput()

Colour input



colourpicker::colourInput()

Date range

2014-01-24 to 2014-01-24

dateRangeInput()

File input

Choose File No file chosen

fileInput()

Numeric input

1

numericInput()

Password Input

••••••••••

passwordInput()

Text area

Multiple lines
of text

textAreaInput()

Radio buttons

Choice 1
Choice 2
Choice 3

radioButtons()

Select box

Choice 1

selectInput()

Sliders

sliderInput()

Text input

Enter text...

textInput()

Adding Input and Control Widgets

- control widgets go in the `sidebarPanel`
- always choose control widgets depending on the design of your app!
- Most common:
  - `radioButtons()`
  - `selectInput()`
  - `sliderInput()`

Adding Input and Control Widgets - Radio Buttons

- We specify the possible values, range and appearance in the control widget

```
radioButtons(
  "buttons",
  "Push a button",
  choices = c("One", "Two", "Three"),
  selected = "One"
)
```

Adding Input and Control Widgets - Select Input

- We specify the possible values, range and appearance in the control widget

```
selectInput(
  "selector",
  "Select a color",
  choices = c("Blue", "Green", "Yellow")
)
```

Adding Input and Control Widgets - Slider Input

- We specify the possible values, range and appearance in the control widget

```
sliderInput(
  "slider",
  "Pick a range",
  min = 0,
  max = 100,
  value = c(10,30),
  pre = "€"
)
```

Exercise I - Building your own UI

- Using the `mtcars` data set, we will now start creating our own Shiny Apps

1. Make yourself familiar with the data set, if you don't know it yet.

```
library(tidyverse)
glimpse(mtcars)
?mtcars
```

2. We want to create a classic data presentation app. Think of an appropriate UI for the data presentation. If you like, you can draw a sketch.
3. Think of useful control widgets to control the presentation of the data.
4. Create the two files needed for Shiny Apps, add the relevant code to initiate the app and set up an UI with a sidebar.
5. Add the control widgets and specify the conditions, you want the users to manipulate
6. Run the app regularly to see how you proceed

Implementing the server logic

Implementing the server logic

XXX