BUILDING WEB APPLICATIONS IN R WITH SHINY: CASE STUDIES

Explore a dataset with Shiny

Dean Attali
Shiny Consultant

Explore a dataset with Shiny

Dataset

+ Interactive environment

+ View data

+ Filter data

+ Download data

= **Shiny app**

Visualize data as a table

country	continent	year	lifeExp	pop	gdpPercap
Afghanistan	Asia	1952	28.801	8425333	779.4453145
Afghanistan	Asia	1957	30.332	9240934	820.8530296
Afghanistan	Asia	1962	31.997	10267083	853.10071
Afghanistan	Asia	1967	34.02	11537966	836.1971382
Afghanistan	Asia	1972	36.088	13079460	739.9811058
Afghanistan	Asia	1977	38.438	14880372	786.11336
Afghanistan	Asia	1982	39.854	12881816	978.0114388
Afghanistan	Asia	1987	40.822	13867957	852.3959448
Afghanistan	Asia	1992	41.674	16317921	649.3413952
Afghanistan	Asia	1997	41.763	22227415	635.341351

Tables in shiny

- Tables are output
- Outputs use output placeholder functions in UI:

```
tableOutput("my_table")
```

- Outputs use render functions in the server:

```
output$my_table <- renderTable({
    gapminder
})
```

Filtering table data

- Inputs can be used to filter
- Add input to UI:

```
selectInput("country", "Country",
            choices = levels(gapminder$country))
```

- Filter data using input:

```
output$my_table <- renderTable({
    subset(gapminder, country == input$country)
})
```

Select input choices

- `choices` argument of `selectInput()` can be any list of strings
- `choices` can be subset of variable

```
selectInput("country", "Country",
            choices = levels(gapminder$country)[1:10])
```

- `choices` can be expanded to add new values

```
selectInput("country", "Country",
            choices = c("any", levels(gapminder$country)))
```

BUILDING WEB APPLICATIONS IN R WITH SHINY: CASE STUDIES

Let’s practice!

BUILDING WEB APPLICATIONS IN R WITH SHINY: CASE STUDIES

More ways to view data: plot and download

Dean Attali
Shiny Consultant

Plot data

- Plots are common first step when exploring new dataset
- Plots are outputs
- Plot output placeholder function in UI:

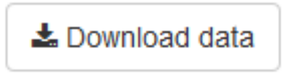
```
plotOutput("my_plot")
```

- Plot render function in the server:

```
output$my_plot <- renderPlot({
    # code for a plot
})
```

Download data

- Downloading is supported using download button



- Can create any type of file to download
  - image files, text files, CSV files

CSV files

- **C**omma **S**eparated **V**alues
- Store small-medium datasets
- CSV of `gapminder`:

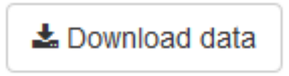
```
country,continent,year,lifeExp,pop,gdpPercap
Afghanistan,Asia,1952,28.801,8425333,779.4453145
Afghanistan,Asia,1957,30.332,9240934,820.8530296
Afghanistan,Asia,1962,31.997,10267083,853.10071
Afghanistan,Asia,1967,34.02,11537966,836.1971382
```

- Create CSV file:

```
write.csv(gapminder, "myfile.csv")
```

Download data in Shiny

- Download button is treated as output



- Add download button to UI:
  (similar to output functions)

```
downloadButton(outputId = “download_data”,
               label = “Download data”)
```

- Add download handler in server:
  (similar to render functions)

```
output$download_data <- downloadHandler(
    filename = "data.csv",
    content = function(file) {
        # Code that creates a file in the path <file>
        write.csv(gapminder, file)
    }
)
```

Download handler

```
output$download_data <- downloadHandler(
    filename = “data.csv”,
    content = function(file) {
        # code that creates a file in the path <file>
        write.csv(gapminder, file)
    }
)
```

- `downloadHandler()` has two arguments
  - **filename**
    Name of downloaded file
  - **content(file)**
    Function with 1 argument
    Create the file to download, argument is file path

BUILDING WEB APPLICATIONS IN R WITH SHINY: CASE STUDIES

Let’s practice!

BUILDING WEB APPLICATIONS IN R WITH SHINY: CASE STUDIES

Reactive variables

Dean Attali
Shiny Consultant

Code duplication

```
data <- gapminder
data <- subset(
    data,
    lifeExp >= input$life[1] & lifeExp <= input$life[2]
)
if (input$continent != "All") {
    data <- subset(
        data,
        continent == input$continent
    )
}
```

Duplicated 3 times

1. `renderTable()`
2. `renderPlot()`
3. `downloadHandler()`

Reactive variables reduce code duplication

- Duplicated code ⇒ multiple places to maintain
    - When code needs updating
    - When bugs need fixing
- Easy to forget one instance, leading to bugs
- Use `reactive()` variables instead of code duplication

Reactive variables

```
output$my_table <- renderTable({
    data <- gapminder
    data <- subset(
        data,
        lifeExp >= input$life[1] & lifeExp <= input$life[2]
    )
})
```

```
my_data <- reactive({
    data <- gapminder
    data <- subset(
        data,
        lifeExp >= input$life[1] & lifeExp <= input$life[2]
    )
})

output$my_table <- renderTable({
    my_data()
})
```

Reactive variables caching

- Reactive variables **cache** their values
- Remember their own value
- Do not run again if dependencies didn't change

Reactive variables caching

```
output$table <- renderTable({
    fit_model(input$num)
})

output$plot <- renderPlot({
    ggplot(
      fit_model(input$num), ...)
})
```

`fit_model()` takes 5s

`fit_model()` called twice = 10s

```
x <- reactive({
    fit_model(input$num)
})

output$table <- renderTable({
    x()
})
output$plot <- renderPlot({
    ggplot(x(), ...)
})
```

`x()` called twice, code inside x runs once

`fit_model()` called once = 5s

Reactive variables are lazy

Lazy variable = not calculated until value is needed

```
x <- reactive({
    fit_model(input$num)
})

output$download <- downloadHandler(
    filename = "x.csv",
    content = function(file) {
        write.csv(x(), file)
    }
)
```

`x()` only runs when download is requested,
not every time `input$num` changes

BUILDING WEB APPLICATIONS IN R WITH SHINY: CASE STUDIES

Let’s practice!

BUILDING WEB APPLICATIONS IN R WITH SHINY: CASE STUDIES

Visual enhancements

Dean Attali
Shiny Consultant

Shiny tables

```
tableOutput("table")
```

```
output$table <- renderTable({ gapminder })
```

country	continent	year	lifeExp	pop	gdpPercap
Afghanistan	Asia	1952	28.80	8425333	779.45
Afghanistan	Asia	1957	30.33	9240934	820.85
Afghanistan	Asia	1962	32.00	10267083	853.10
Afghanistan	Asia	1967	34.02	11537966	836.20
Afghanistan	Asia	1972	36.09	13079460	739.98
Afghanistan	Asia	1977	38.44	14880372	786.11

Make better tables with DT

```
DT::dataTableOutput("table")
```

```
output$table <- DT::renderDataTable({ gapminder })
```

Show 10 entries Search:

	country	continent	year	lifeExp	pop	gdpPercap
1	Afghanistan	Asia	1952	28.801	8425333	779.4453145
2	Afghanistan	Asia	1957	30.332	9240934	820.8530296
3	Afghanistan	Asia	1962	31.997	10267083	853.10071
4	Afghanistan	Asia	1967	34.02	11537966	836.1971382
5	Afghanistan	Asia	1972	36.088	13079460	739.9811058
6	Afghanistan	Asia	1977	38.438	14880372	786.11336
7	Afghanistan	Asia	1982	39.854	12881816	978.0114388
8	Afghanistan	Asia	1987	40.822	13867957	852.3959448
9	Afghanistan	Asia	1992	41.674	16317921	649.3413952
10	Afghanistan	Asia	1997	41.763	22227415	635.341351

Showing 1 to 10 of 1,704 entries Previous 1 2 3 4 5 … 171 Next

Split the UI into tabs

Parameters | Plot | Table | About this app

Gapminder data

Show 10 entries Search:

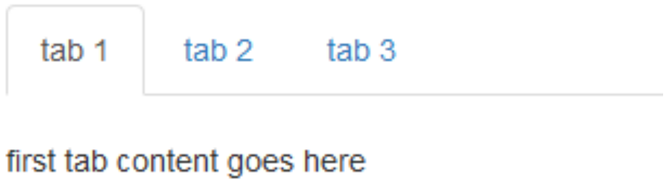
	country	continent	year	lifeExp	pop	gdpPercap
1	Afghanistan	Asia	1952	28.801	8425333	779.4453145
2	Afghanistan	Asia	1957	30.332	9240934	820.8530296
3	Afghanistan	Asia	1962	31.997	10267083	853.10071
4	Afghanistan	Asia	1967	34.02	11537966	836.1971382
5	Afghanistan	Asia	1972	36.088	13079460	739.9811058
6	Afghanistan	Asia	1977	38.438	14880372	786.11336
7	Afghanistan	Asia	1982	39.854	12881816	978.0114388
8	Afghanistan	Asia	1987	40.822	13867957	852.3959448
9	Afghanistan	Asia	1992	41.674	16317921	649.3413952
10	Afghanistan	Asia	1997	41.763	22227415	635.341351

Showing 1 to 10 of 1,704 entries Previous 1 2 3 4 5 … 171 Next

Split the UI into tabs

```
tabPanel(title = "tab 1", "content goes here")
tabPanel(title = "tab 2", "second tab", plotOutput("plot"))
```

```
fluidPage(
  tabsetPanel(
    tabPanel(title = "tab 1", "first tab content goes here"),
    tabPanel(title = "tab 2", "second tab", plotOutput("plot")),
    tabPanel(title = "tab 3", textInput("text", "Name", ""))
  )
)
```



CSS: Fine-tune your app's look

- **C**ascading **S**tyle **S**heets
- Markup language to customize look of any element in webpage
    - Shiny UI is a webpage
- Background colour, text colour, text size, whitespace, fonts, ...

CSS syntax

- CSS rules syntax

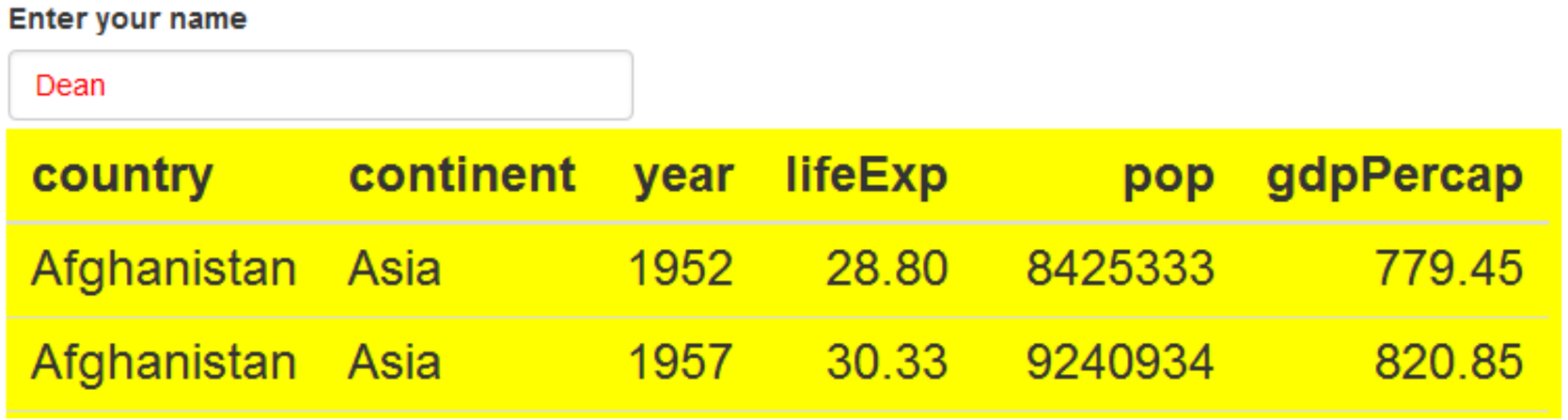
```
#ID {
  property: value;
  property: value;
  ...
}
```

- `ID` is element ID to apply the style to
- To add CSS to Shiny, use `tags$style()`

```
ui <- fluidPage(
  tags$style("
    #ID {
      property: value;
    }
  ")
)
```

CSS example

```
ui <- fluidPage(
  textInput("name", "Enter your name", "Dean"),
  tableOutput("table")
)
```



CSS example

```
css <- "
  #name {
    color: red;
  }
  #table {
    background: yellow;
    font-size: 24px;
  }
"

ui <- fluidPage(
  tags$style(css),
  textInput("name", "Enter your name", "Dean"),
  tableOutput("table")
)
```

BUILDING WEB APPLICATIONS IN R WITH SHINY: CASE STUDIES

Let’s practice!