Kostenloses eBook

LERNEN shiny

Free unaffiliated eBook created from
Stack Overflow contributors.

#shiny

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: shiny

It is an unofficial and free shiny ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official shiny.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Shiny

Bemerkungen

In diesem Abschnitt erhalten Sie einen Überblick, was glänzend ist und warum ein Entwickler es verwenden möchte.

Es sollte auch alle großen Themen innerhalb von glänzend erwähnen und auf verwandte Themen verweisen. Da die Dokumentation für Shiny neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

Examples

Installation oder Setup

Shiny kann als eigenständige Anwendung auf Ihrem lokalen Computer ausgeführt werden, auf einem Server, der glänzende Apps für mehrere Benutzer bereitstellen kann (mithilfe von Shiny Server) oder auf shinyapps.io .

1. **Installieren** von **Shiny auf einem lokalen Computer:** in R / RStudio, lief `install.packages("shiny")` , wenn von CRAN installieren oder `devtools::install_github("rstudio/shiny")` , wenn aus dem Repository RStudio Github installieren. Das Github-Repository beherbergt eine Entwicklungsversion von Shiny, die im Vergleich zur CRAN-Version möglicherweise mehr Funktionen bietet, aber auch instabil sein kann.

Wann würde ich glänzend verwenden?

1. Ich habe einige Datenanalysen für einige Daten und viele nichtcodierende Personen im Team, die ähnliche Daten wie meine haben und ähnliche Analyseanforderungen haben. In solchen Fällen kann ich eine Webanwendung mit Shiny erstellen, die benutzerspezifische Eingabedatendateien aufnimmt und Analysen generiert.
2. Ich muss analysierte Daten oder relevante Diagramme mit anderen im Team teilen. Glänzende Web-Apps können in solchen Situationen hilfreich sein.
3. Ich habe keine nennenswerten Erfahrungen mit der Programmierung von Webanwendungen, muss aber schnell eine einfache Schnittstelle zusammenstellen. Zur Rettung glänzend mit einfachen UI- und Server-Elementen und minimaler Codierung.
4. Mit interaktiven Elementen können Ihre Benutzer herausfinden, welches Element der Daten für sie relevant ist. Sie könnten beispielsweise Daten für das gesamte Unternehmen geladen haben, aber pro Abteilung eine Dropdown-Liste wie "Vertrieb", "Produktion", "Finanzen" haben, in der die Daten so zusammengefasst werden können, wie sie die Benutzer anzeigen möchten. Die Alternative wäre ein umfangreiches Berichtspaket mit Analysen für jede Abteilung, aber sie lesen nur ihr Kapitel und die Gesamtzahl.

Einfache App

Jede `shiny` App enthält zwei Teile: Eine Benutzeroberflächendefinition (`UI`) und ein Serverskript (`server`). Dieses Beispiel zeigt, wie Sie "Hallo Welt" von der Benutzeroberfläche oder vom Server aus drucken können.

UI.R

In der Benutzeroberfläche können Sie einige Ansichtsobjekte (div, Eingaben, Schaltflächen usw.) platzieren.

```
library(shiny)

# Define UI for application print "Hello world"
shinyUI(

  # Create bootstrap page
  fluidPage(

    # Paragraph "Hello world"
    p("Hello world"),

    # Create button to print "Hello world" from server
    actionButton(inputId = "Print_Hello", label = "Print_Hello World"),

    # Create position for server side text
    textOutput("Server_Hello")

  )
)
```

Server.R

Im Serverskript können Sie Methoden definieren, mit denen Daten bearbeitet oder Aktionen überwacht werden.

```
# Define server logic required to print "Hello World" when button is clicked
shinyServer(function(input, output) {

  # Create action when actionButton is clicked
  observeEvent(input$Print_Hello,{

    # Change text of Server_Hello
    output$Server_Hello = renderText("Hello world from server side")
  })


})
```

Wie man läuft

Sie können Ihre App auf verschiedene Arten ausführen:

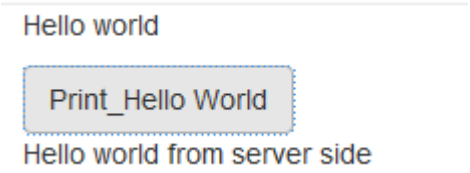
1. Erstellen Sie zwei verschiedene Dateien und legen Sie sie in einem Verzeichnis ab. Verwenden `runApp('your dir path')` dann `runApp('your dir path')`

2. Sie können zwei Variablen definieren (z. B. `shinyApp(ui,server)` und Server) und anschließend `shinyApp(ui,server)` zum Ausführen Ihrer App verwenden

Ergebnis

In diesem Beispiel sehen Sie Text und eine Schaltfläche:

Und nach dem Klicken der Schaltfläche antwortet der Server:



Einschließlich Grundstücke

Die einfachste Möglichkeit, Plots in Ihre shinyApp aufzunehmen, besteht darin, `plotOutput` in der `renderPlot` und `renderPlot` im Server zu verwenden. Dies funktioniert sowohl mit `ggPlot` als auch mit `ggPlot`

```
library(shiny)
library(ggplot2)

ui <- fluidPage(
  plotOutput('myPlot'),
  plotOutput('myGgPlot')
)

server <- function(input, output, session){
  output$myPlot = renderPlot({
    hist(rnorm(1000))
  })
  output$myGgPlot <- renderPlot({
    ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) + geom_point()
  })
}

shinyApp(ui, server)
```

Einschließlich Tabellen

Tabellen sind am einfachsten im DT-Paket enthalten , bei dem es sich um eine R-Schnittstelle zur JavaScript-Bibliothek DataTables handelt.

```
library(shiny)
library(DT)

ui <- fluidPage(
  dataTableOutput('myTable')
)

server <- function(input, output, session){
  output$myTable <- renderDataTable({
    datatable(iris)
  })
}

shinyApp(ui, server)
```

Erste Schritte mit Shiny online lesen: https://riptutorial.com/de/shiny/topic/2667/erste-schritte-mit-shiny

Kapitel 2: Daten in glänzend hochladen

Examples

.RData-Dateien mit fileInput () in Shiny hochladen

In diesem Beispiel können Sie .RData-Dateien hochladen. Beim Ansatz mit `load` und `get` können Sie die geladenen Daten einem Variablennamen Ihrer Wahl zuweisen. Für das Beispiel "Standalone" habe ich den oberen Abschnitt eingefügt, der zwei Vektoren auf Ihrer Festplatte speichert, um sie später zu laden und zu zeichnen.

```
library(shiny)

# Define two datasets and store them to disk
x <- rnorm(100)
save(x, file = "x.RData")
rm(x)
y <- rnorm(100, mean = 2)
save(y, file = "y.RData")
rm(y)

# Define UI
ui <- shinyUI(fluidPage(
  titlePanel(".RData File Upload Test"),
  mainPanel(
    fileInput("file", label = ""),
    actionButton(inputId="plot","Plot"),
    plotOutput("hist"))
  )
)

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

  observeEvent(input$plot,{
    if ( is.null(input$file)) return(NULL)
    inFile <- input$file
    file <- inFile$datapath
    # load the file into new environment and get it from there
    e = new.env()
    name <- load(file, envir = e)
    data <- e[[name]]

    # Plot the data
    output$hist <- renderPlot({
      hist(data)
    })
  })
})

# Run the application
shinyApp(ui = ui, server = server)
```

Hochladen von CSV-Dateien in Shiny

Es ist auch möglich, dass ein Benutzer csvs in Ihre Shiny-App hochlädt. Der folgende Code zeigt ein kleines Beispiel, wie dies erreicht werden kann. Es enthält auch einen radioButton-Eingang, sodass der Benutzer interaktiv das zu verwendende Trennzeichen auswählen kann.

```
library(shiny)
library(DT)

# Define UI
ui <- shinyUI(fluidPage(

  fileInput('target_upload', 'Choose file to upload',
            accept = c(
              'text/csv',
              'text/comma-separated-values',
              '.csv'
            )),
  radioButtons("separator","Separator: ",choices = c(";",",","","":"), selected=";",inline=TRUE),
  DT::dataTableOutput("sample_table")
)
)

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

  df_products_upload <- reactive({
    inFile <- input$target_upload
    if (is.null(inFile))
      return(NULL)
    df <- read.csv(inFile$datapath, header = TRUE,sep = input$separator)
    return(df)
  })

  output$sample_table<- DT::renderDataTable({
    df <- df_products_upload()
    DT::datatable(df)
  })

}
)

# Run the application
shinyApp(ui = ui, server = server)
```

Daten in glänzend hochladen online lesen: https://riptutorial.com/de/shiny/topic/7576/daten-in-glanzend-hochladen

Kapitel 3: Javascript API

Syntax

- Sitzung $ sendCustomMessage (*Name* , *Liste der Parameter*)
- Shiny.addCustomMessageHandler (*Name* , *JS-Funktion, die eine Liste von Parametern akzeptiert*)
- Shiny.onInputChange (*Name* , *Wert*)

Examples

Daten werden vom Server an den Client gesendet

In vielen Fällen möchten Sie Daten vom R-Server an den JS-Client senden. Hier ist ein sehr einfaches Beispiel:

```
library(shiny)
runApp(
  list(
    ui = fluidPage(
      tags$script(
        "Shiny.addCustomMessageHandler('message', function(params) { alert(params); });"
      ),
      actionButton("btn","Press Me")
    ),
    server = function(input, output, session) {
      observeEvent(input$btn,{
        randomNumber <- runif(1,0,100)
        session$sendCustomMessage("message",list(paste0(randomNumber," is a random number!")))
      })
    }
  )
)
```

Die Arbeitspferde hier sind die `session$sendCustomMessage` Funktion in `R` und die `Shiny.addCustomMessageHandler` Funktion in `javascript` .

Mit der `session$sendCustomMessage` Funktion können Sie Parameter von `R` an eine `javascript` Funktion senden. Mit `Shiny.addCustomMessageHandler` Sie die `javascript` Funktion definieren, die die Parameter von `R` akzeptiert.

Hinweis: Listen werden in JSON konvertiert, wenn sie von `R` an `javascript`

Daten werden vom Client an den Server gesendet

In einigen Fällen möchten Sie Daten vom JS-Client an den R-Server senden. Hier ein einfaches Beispiel für die Verwendung der `Shiny.onInputChange` Funktion von Javascript:

```
library(shiny)
```

```
runApp(
  list(
    ui = fluidPage(
      # create password input
      HTML('<input type="password" id="passwordInput">'),
      # use jquery to write function that sends value to
      # server when changed
      tags$script(
        '$("#passwordInput").on("change",function() {
          Shiny.onInputChange("myInput",this.value);
        })'
      ),
      # show password
      verbatimTextOutput("test")
    ),
    server = function(input, output, session) {
      # read in then show password
      output$test <- renderPrint(
        input$myInput
      )
    }
  )
)
```

Hier erstellen wir eine Passworteingabe mit der ID `passwordInput` . Wir fügen auf der Benutzeroberfläche eine Javascript-Funktion hinzu, die auf Änderungen in `passwordInput` reagiert und den Wert mithilfe von `Shiny.onInputChange` an den Server `Shiny.onInputChange` .

`Shiny.onInputChange` benötigt zwei Parameter, einen Namen für die `input$*name*` und einen Wert für die `input$*name*`

Dann können Sie die `input$*name*` wie jede andere Shiny-Eingabe verwenden.

Javascript API online lesen: https://riptutorial.com/de/shiny/topic/3149/javascript-api

Kapitel 4: reaktiv, reaktivWert und eventReaktiv, beobachten und beobachtenEvent in Shiny

Einführung

reaktive, reaktiveWerte und eventReactive sind verschiedene Arten von reaktiven Ausdrücken in Shiny. Sie liefern eine Ausgabe, die als Eingabe in andere Ausdrücke verwendet werden kann, die wiederum eine Abhängigkeit vom reaktiven Ausdruck haben.

observ und observEvent ähneln reaktiven Ausdrücken. Der große Unterschied ist, dass die Beobachter keine Leistung bringen und daher nur für ihre Nebenwirkungen nützlich sind.

Beispiele für ihre Verwendung finden Sie in diesem Dokument.

Examples

reaktiv

Ein reaktives Element kann verwendet werden, um die Ausgabe von einem anderen Ausdruck abhängig zu machen. Im folgenden Beispiel ist das Ausgabeelement $ text von text_reactive abhängig, das wiederum von der Eingabe $ user_text abhängt. Immer wenn sich die Eingabe $ user_text ändert, werden die Ausgabe $ text element und text_reactive **ungültig.** Sie werden basierend auf dem neuen Wert für die Eingabe $ user_text neu berechnet.

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example reactive"),

  mainPanel(

    # input field
    textInput("user_text", label = "Enter some text:", placeholder = "Please enter some
text."),

    # display text output
    textOutput("text"))
)

server <- function(input, output) {

  # reactive expression
  text_reactive <- reactive({
    input$user_text
  })

  # text output
```

```
  output$text <- renderText({
    text_reactive()
  })
}

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

eventReactive

eventReactives ähneln Reaktiven und sind wie folgt aufgebaut:

```
eventReactive( event {
code to run
})
```

eventReactives sind nicht von allen reaktiven Ausdrücken in ihrem Körper abhängig (" *Code to run* " im obigen Snippet). Sie sind stattdessen nur von den im *Ereignisabschnitt* angegebenen Ausdrücken abhängig.

In dem folgenden Beispiel haben wir eine Schaltfläche zum Senden hinzugefügt und eine eventReactive erstellt. Wenn sich die Eingabe $ user_text ändert, wird eventReactive nicht ungültig gemacht, da eventReactive nur von der actionButton-Eingabe $ submit abhängig ist. Immer wenn diese Schaltfläche gedrückt wird, werden text_reactive und nachfolgende Ausgabe von $ text ungültig und werden basierend auf der aktualisierten Eingabe $ user_text neu berechnet.

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example eventReactive"),

  mainPanel(

    # input field
    textInput("user_text", label = "Enter some text:", placeholder = "Please enter some
text."),

    # submit button
    actionButton("submit", label = "Submit"),

    # display text output
    textOutput("text"))
)

server <- function(input, output) {

  # reactive expression
  text_reactive <- eventReactive( input$submit, {
    input$user_text
  })

  # text output
  output$text <- renderText({
    text_reactive()
  })
```

```
}

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

reaktive Werte

Der Wert vonactiveValues kann zum Speichern von Objekten verwendet werden, von denen andere Ausdrücke abhängig sein können.

Im folgenden Beispiel wird ein interactiveValues-Objekt mit dem Wert "Es wurde noch kein Text gesendet." Initialisiert. Ein separater Beobachter wird erstellt, um das ReactiveValues-Objekt zu aktualisieren, wenn Sie auf die Schaltfläche "Senden" klicken. Beachten Sie, dass die reaktiven Werte selbst nicht von den Ausdrücken in ihrem Körper abhängig sind.

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example reactiveValues"),

  mainPanel(

    # input field
    textInput("user_text", label = "Enter some text:", placeholder = "Please enter some
text."),
    actionButton("submit", label = "Submit"),

    # display text output
    textOutput("text"))
)

server <- function(input, output) {

  # observe event for updating the reactiveValues
  observeEvent(input$submit,
               {
    text_reactive$text <- input$user_text
  })

  # reactiveValues
  text_reactive <- reactiveValues(
    text = "No text has been submitted yet."
  )

  # text output
  output$text <- renderText({
    text_reactive$text
  })
}

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

observEvent

Ein observEvent-Objekt kann verwendet werden, um bei Auftreten eines bestimmten Ereignisses einen Code auszulösen. Es ist aufgebaut als:

```
observeEvent( event {
code to run
})
```

Das observEvent hängt nur vom Abschnitt *"event"* in dem kleinen Code ab. Es ist nicht von etwas im " *Code to run* " -Teil abhängig. Eine Beispielimplementierung finden Sie unten:

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example reactive"),

  mainPanel(

    # action buttons
    actionButton("button1","Button 1"),
    actionButton("button2","Button 2")
  )
)

server <- function(input, output) {

  # observe button 1 press.
  observeEvent(input$button1, {
    # The observeEvent takes no dependency on button 2, even though we refer to the input in
the following line.
    input$button2
    showModal(modalDialog(
      title = "Button pressed",
      "You pressed one of the buttons!"
    ))
  })
}

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

beobachten

Ein Beobachtungsausdruck wird jedes Mal ausgelöst, wenn sich einer seiner Eingänge ändert. Der Hauptunterschied in Bezug auf einen reaktiven Ausdruck besteht darin, dass er keine Ausgabe liefert und nur für seine Nebenwirkungen verwendet werden sollte (z. B. das Ändern eines ReactiveValues-Objekts oder das Auslösen eines Popups).

Beachten Sie auch, dass observ NULL nicht ignoriert. Daher wird auch dann ausgelöst, wenn die Eingänge noch NULL sind. ObservEvent ignoriert standardmäßig NULL, da dies fast immer wünschenswert ist.

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example reactive"),

  mainPanel(

    # action buttons
```

```
    actionButton("button1","Button 1"),
    actionButton("button2","Button 2")
  )
)

server <- function(input, output) {

  # observe button 1 press.
  observe({
    input$button1
    input$button2
    showModal(modalDialog(
      title = "Button pressed",
      "You pressed one of the buttons!"
    ))
  })
}

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

reaktiv, reaktivWert und eventReaktiv, beobachten und beobachtenEvent in Shiny online lesen: https://riptutorial.com/de/shiny/topic/10787/reaktiv--reaktivwert-und-eventreaktiv--beobachten-und-beobachtenevent-in-shiny

Kapitel 5: Wie schreibe ich MCVE (Minimales, vollständiges und überprüfbares Beispiel)? Glänzende Apps

Einführung

Wenn Sie Probleme mit Ihren Shiny-Apps haben, sollten Sie eine App erstellen, die Ihren Standpunkt veranschaulicht. Diese App sollte so einfach wie möglich sein und gleichzeitig Ihr Problem widerspiegeln. Dies bedeutet, einfache Datensätze zu verwenden, selbsterklärende Benennung (insbesondere für E / A-IDs) und Plots durch einfachere zu ersetzen.

Es ist auch ratsam, Ihren MCVE so zu erstellen, dass möglichst wenig Nicht-Standard-Bibliotheken erforderlich sind.

Examples

Grundstruktur

MCVEs sollten die Shiny-App starten, wenn sie in die Konsole kopiert werden. Ein einfacher Weg, dies zu tun, ist die Verwendung der `shinyApp` Funktion. Zum Beispiel:

> Warum reagiert mein Kontrollkästchen nicht?

```
library(shiny)

ui <- fluidPage(
  checkboxInput('checkbox', 'click me'),
  verbatimTextOutput('text')
)

server <- function(input, output, session){
  output$text <- renderText({
    isolate(input$checkbox)
  })
}

shinyApp(ui, server)
```

Alternativ können Sie der `ui` und dem `server` auch keine Variablen zuweisen.

```
library(shiny)

shinyApp(
  fluidPage(
    checkboxInput('checkbox', 'click me'),
    verbatimTextOutput('text')
  ),
```

```
  function(input, output, session){
    output$text <- renderText({
      isolate(input$checkbox)
    })
  }
)

shinyApp(ui, server)
```

Vermeiden Sie unnötige Details

In der Praxis sind glänzende Apps oft sehr kompliziert und voller Funktionen, die im Laufe der Zeit entwickelt wurden. Meistens sind diese zusätzlichen Details nicht erforderlich, um Ihr Problem zu reproduzieren. Es ist am besten, wenn Sie solche Details beim Schreiben von MCVE überspringen.

FALSCH

> Warum wird meine Handlung nicht angezeigt?

```
library(shiny)
library(ggplot2)

ui <- fluidPage(
  plotOutput('plot')
)

server <- function(input, output, session){
  df <- data.frame(treatment = rep(letters[1:3], times = 3),
                   context = rep(LETTERS[1:3], each = 3),
                   effect = runif(9,0,1))
  df$treat.con <- paste(df$treatment,df$context, sep = ".")
  df$treat.con <- reorder(df$treat.con, -df$effect, )
  output$plot = renderPlot({
    myPlot <- ggplot(df, aes(x = treat.con, y = effect)) +
       geom_point() +
       facet_wrap(~context,
                  scales = "free_x",
                  ncol = 1)
  })
}

shinyApp(ui, server)
```

RECHT

> Warum wird mein Plot nicht angezeigt?

```
library(shiny)
library(ggplot2)

ui <- fluidPage(
```

```
  plotOutput('plot')
)

server <- function(input, output, session){
  output$plot = renderPlot({
    myPlot <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
  })
}

shinyApp(ui, server)
```

Wie schreibe ich MCVE (Minimales, vollständiges und überprüfbares Beispiel)? Glänzende Apps online lesen: https://riptutorial.com/de/shiny/topic/10653/wie-schreibe-ich-mcve--minimales--vollstandiges-und-uberprufbares-beispiel---glanzende-apps

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Shiny	Batanichek, Bogdan Rau, chrki, Community, Florian, Gregor de Cillia, jsb, Mathias711, micstr, sigmabeta
2	Daten in glänzend hochladen	Florian, symbolrush
3	Javascript API	Carl, Tomás Barcellos
4	reaktiv, reaktivWert und eventReaktiv, beobachten und beobachtenEvent in Shiny	Florian
5	Wie schreibe ich MCVE (Minimales, vollständiges und überprüfbares Beispiel)? Glänzende Apps	Florian, Gregor de Cillia