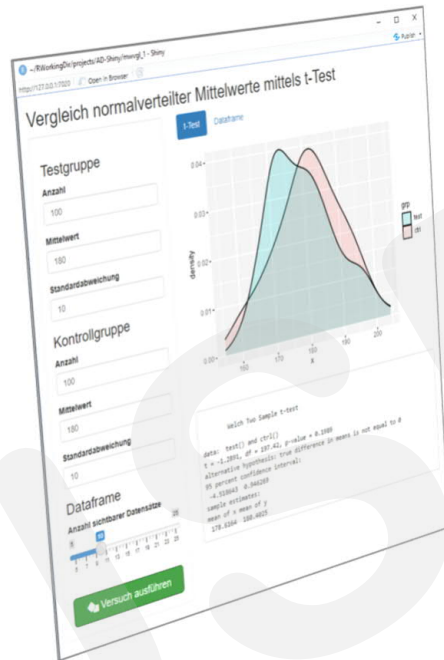


# Shiny R in 30 Minuten



HTML

CSS

JavaScript

Bootstrap



Statistiker - KI-Experte - R/Python-Spezialist



***Ihr Trainer für  
professionelle Datenanalyse***

***Statistiker, KI-Experte und R/Python-Spezialist***



***Helmut Grillenberger***

# Shiny Cheatsheet (Spickzettel)

### Interactive Web Apps with shiny Cheat Sheet

learn more at [shiny.rstudio.com](http://shiny.rstudio.com)

#### Studio

#### Basics

A Shiny app is a web page (UI) connected to a computer running a live R session (Server).

Users can manipulate the UI, which will cause the server to update the UI's displays. (By running R code).

#### App template

```
library(shiny)
ui <- fluidPage()
server <- function(input, output){
  shinyApp(ui = ui, server = server)
}
```

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines **ui** and **server** into a functioning app. Wrap with **runApp()** if calling from a sourced script or inside a function.

#### Share your app

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at <http://shinyapps.io>
2. Click the **Publish** icon in the RStudio IDE (>=0.99) or run:
 

```
rconect::deployApp("~/path to directory")
```

Build or purchase your own Shiny Server at [www.rstudio.com/products/shiny-server/](http://www.rstudio.com/products/shiny-server/)

### Building an App

Complete the template by adding arguments to fluidPage() and a body to the server function.

Add inputs to the UI with **"Input"** functions. Add outputs with **"Output"** functions. Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with **output\$**id.
2. Refer to inputs with **input\$**id.
3. Wrap code in a **render()** function before saving to output.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
  shinyApp(ui = ui, server = server)
}
```

Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
ui.R
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server.R
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

**ui.R** contains everything you would save to ui.

**server.R** ends with the function you would save to server.

No need to call shinyApp()

Save each app as a directory that contains an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

- **app.R** - The directory name is the name of the app (optional) defines objects available to both ui.R and server.R
- **global.R** - (optional) used in showcase mode
- **DESCRIPTION** - (optional) data, scripts, etc.
- **README** - (optional) directory of files to share with web browsers (images, CSS, js, etc.) Must be named "www"
- **<other files>** - (optional) directory of files to share with web browsers (images, CSS, js, etc.) Must be named "www"

Launch apps with **runApp()** (path to directory)

### Outputs - render() and \*Output() functions work together to add R output to the UI

<b>renderDataTable</b> (expr, quoted, escape, env, quoted)	<b>dataTableOutput</b> (outputId, icon, ...)
<b>renderImage</b> (expr, env, quoted, deleteFile)	<b>imageOutput</b> (outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, click, hover, inline)
<b>renderPlot</b> (expr, width, height, res, ... env, quoted, inline)	<b>plotOutput</b> (outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, click, hover, inline)
<b>renderPrint</b> (expr, env, quoted, func, width)	<b>verbatimTextOutput</b> (outputId)
<b>renderTable</b> (expr, ... env, quoted, func)	<b>tableOutput</b> (outputId)
<b>renderText</b> (expr, env, quoted, func)	<b>textOutput</b> (outputId, container, inline)
<b>renderUI</b> (expr, env, quoted, func)	<b>uiOutput</b> (outputId, inline, container, ...)

### Inputs - collect values from the user

Access the current value of an input object with **input\$**inputId. Input values are reactive.

Action: **actionButton**(inputId, label, icon, ...)

Link: **actionLink**(inputId, label, icon, ...)

Choice 1: **checkboxGroupInput**(inputId, label, choices, selected, inline)

Choice 2: **checkboxGroupInput**(inputId, label, value)

Choice 3: **checkboxGroupInput**(inputId, label, value)

Check me: **checkboxGroupInput**(inputId, label, value)

**dateInput**(inputId, label, value, min, max, format, startview, weekstart, language)

**dateRangeInput**(inputId, label, start, end, min, format, startview, weekstart, language, separator)

**fileInput**(inputId, label, multiple, accept)

**numericInput**(inputId, label, value, min, max, step)

**passwordInput**(inputId, label, value)

Choice A: **radioButtons**(inputId, label, choices, selected, inline)

Choice B: **radioButtons**(inputId, label, choices, selected, inline)

Choice C: **radioButtons**(inputId, label, choices, selected, multiple, selectize, width, size) also **selectInput()**

Choice 1: **sliderInput**(inputId, label, min, max, value, step, format, locale, ticks, animate, width, sep, pre, post)

Choice 2: **sliderInput**(inputId, label, min, max, value, step, format, locale, ticks, animate, width, sep, pre, post)

Apply Changes: **submitButton**(text, icon) (triggers reactions across entire app)

Enter text: **textInput**(inputId, label, value)

### Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error **Operation not allowed without an active reactive context**.

Trigger arbitrary code (observeEvent(), observe())

Modularize reactions (reactive())

Prevent reactions (isolate())

Delay reactions (eventReactive())

Render reactive output (render())

inputs x → expression() → outputs y

Create your own reactive values (reactiveValues(), input())

Update

### Create your own reactive values

```
library(shiny)
ui <- fluidPage(
  textInput("x", "")
)
server <- function(input, output) {
  reactiveValues({
    x = input$x
  })
  shinyApp(ui, server)
}
```

**"Input"** functions (see front page)

Each input function creates a reactive value stored as **input\$**inputId-**reactiveValues()** creates a list of reactive values whose values you can set.

### Render reactive output

```
library(shiny)
ui <- fluidPage(
  textInput("x", "")
)
server <- function(input, output) {
  output$y <- renderText({
    input$x + 1
  })
  shinyApp(ui, server)
}
```

Builds an object to display. Will return code in body to rebuild the object whenever a reactive value in the code changes.

Save the results to **output\$**outputId.

### Prevent reactions

```
library(shiny)
ui <- fluidPage(
  textInput("x", "")
)
server <- function(input, output) {
  isolate({
    input$x + 1
  })
  shinyApp(ui, server)
}
```

**isolate(expr)** Runs a code block. Returns a non-reactive copy of the results.

### Trigger arbitrary code

```
library(shiny)
ui <- fluidPage(
  textInput("x", "")
)
server <- function(input, output) {
  observeEvent(input$x, {
    print(input$x)
  })
  shinyApp(ui, server)
}
```

**observeEvent(eventExpr, handlerExpr, env, quoted, label, suspended, priority, domain, autocancel, ignoreNULL)** Suspends code in 2nd argument when reactive values in 1st argument change. See **observe()** for alternative.

### Modularize reactions

```
library(shiny)
ui <- fluidPage(
  textInput("x", "")
)
server <- function(input, output) {
  reactive({
    input$x + 1
  })
  shinyApp(ui, server)
}
```

**reactive(x, env, quoted, label, domain)** Creates a reactive expression that caches its value to reduce computation. Can be called by other code. Notifies its dependencies when it has been invalidated. Call the expression with function syntax, e.g. **f()**.

### Delay reactions

```
library(shiny)
ui <- fluidPage(
  textInput("x", "")
)
server <- function(input, output) {
  eventReactive(input$x, {
    input$x + 1
  })
  shinyApp(ui, server)
}
```

**eventReactive(eventExpr, valueExpr, env, quoted, value, value, quoted, label, domain, ignoreNULL)** Creates reactive expression with code in 2nd argument that only invalidates when reactive values in 1st argument change.

### UI

An app's UI is an HTML document. Use Shiny's functions to assemble this HTML with R.

```
fluidPage(
  textInput("x", "")
)
Returns HTML
<div class="container-fluid">
  <div class="form-group">
    <input type="text" value="" />
  </div>
</div>
```

Add static HTML elements with tags, a list of functions that parallel common HTML tags, e.g. **tags\$a()**. Unnamed arguments will be passed into the tag; named arguments will become tag attributes.

The most common tags we use wrapper functions. You do not need to prefix their names with **tags\$**.

### Header 1

```
ui <- fluidPage(
  h1("Header 1"),
  p("Content")
)
HTML("<div class='container-fluid'>
  <h1>Header 1</h1>
  <p>Content</p>
</div>")
```

To include a CSS file, use **includeCSS()**, or **includeJavaScript()**.

To include JavaScript, use **includeScript()**.

1. Place the file in the **www** subdirectory

2. Link to it with **tags\$head(tags\$link(rel = "stylesheet", type = "text/css", href = "<file name>"))**

1. Place the file in the **www** subdirectory

2. Link to it with **tags\$script(src = "<file name>")**

To include an image

1. Place the file in the **www** subdirectory

2. Link to it with **img(src = "<file name>")**

### Layouts

Combine multiple elements into a "single element" that has its own properties with a panel function, e.g. **wellPanel()**.

```
wellPanel(
  stateInput("s", "m"),
  submitButton()
)
```

Organize panels and elements into a layout with a layout function. Add elements as arguments of the layout functions.

**fluidRow()**

```
ui <- fluidPage(
  fluidRow(
    column(
      fluidRow(
        column(width = 4),
        fluidRow(
          column(width = 2, offset = 3),
          fluidRow(
            column(width = 12)
          )
        )
      )
    )
  )
)
```

**flowLayout()**

```
ui <- fluidPage(
  flowLayout(
    object 1,
    object 2,
    object 3
  )
)
```

**sidebarLayout()**

```
ui <- fluidPage(
  sidebarLayout(
    mainPanel(),
    sidebarPanel()
  )
)
```

**splitLayout()**

```
ui <- fluidPage(
  splitLayout(
    object 1,
    object 2
  )
)
```

**verticalLayout()**

```
ui <- fluidPage(
  verticalLayout(
    object 1,
    object 2,
    object 3
  )
)
```

Layer tabPanels on top of each other, and navigate between them, with:

```
ui <- fluidPage(
  tabsetPanel(
    tabPanel("tab 1", contents()),
    tabPanel("tab 2", contents()),
    tabPanel("tab 3", contents())
  )
)
```

```
ui <- fluidPage(
  navlistPanel(
    tabPanel("tab 1", contents()),
    tabPanel("tab 2", contents()),
    tabPanel("tab 3", contents())
  )
)
```

```
ui <- fluidPage(
  Page(
    tabPanel("tab 1", contents()),
    tabPanel("tab 2", contents()),
    tabPanel("tab 3", contents())
  )
)
```


<https://shiny.rstudio.com/images/shiny-cheatsheet.pdf>

© 2021 USEDATA – Mag. Helmut Grillenberger

3

## Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)



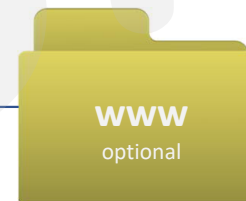
Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

### App template

Begin writing a new app with this template. Preview the app by running the code at the R command line.

```
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)
```

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines **ui** and **server** into a functioning app. Wrap with **runApp()** if calling from a sourced script or inside a function.




shinyapp + SHIFT + TAB

```
library(shiny)
ui <- fluidPage()
server <- function(input, output, session) {}
shinyApp(ui = ui, server = server)
```

css Dateien  
javaScript Dateien  
images (png, jpg ...)

```
runApp( app-name ) → runApp("ttest")
```



# Shiny R Demo



# Shiny App Demo - Skizze

Ansicht: Register 1 (t-Test) `tabPanel "t-Test"`

**input** (indicated by blue arrows pointing to input fields):

- Testgruppe: Anzahl (100, **n1**), Mittelwert (180, **mean1**), Standardabweichung (10, **sd1**)
- Kontrollgruppe: Anzahl (100, **n2**), Mittelwert (180, **mean2**), Standardabweichung (10, **sd2**)
- Dataframe: Anzahl sichtbarer Datensätze (10, **rows**)
- run** button: Versuch ausführen

**output** (indicated by blue arrows pointing to plot and text):

- hist**: Density plot of the data.
- ttest**: Text output of the Welch Two Sample t-test results.

Layout components: `titlePanel`, `tabsetPanel` (t-Test selected), `sidebarPanel`, `mainPanel`.

Ansicht: Register 2 (Dataframe) `tabPanel "Dataframe"`

**output** (indicated by blue arrows pointing to data tables):

- table\_test**: Data for the test group.
- table\_ctrl**: Data for the control group.

Layout components: `titlePanel`, `tabsetPanel` (Dataframe selected), `sidebarPanel`, `mainPanel`. Dimensions: `fluidRow`, `12 Einheiten (EH)`, `column(3 EH)`.

# Danke für Ihr Interesse

Ihr Helmut Grillenberger



Festnetz: **+43 6274 20804**  
Mobile: **+43 664 4212247**  
E-Mail: [helmut.grillenberger@usedata.com](mailto:helmut.grillenberger@usedata.com)  
URL: <https://www.usedata.com>

Inhalt © 2023 USEDATA - Mag. Helmut Grillenberger  
Cartoons wurden über clipart.com lizenziert

