

How to create applications using Shiny

ISGlobal

Barcelona, February 13-14th 2018

Part II: Form design

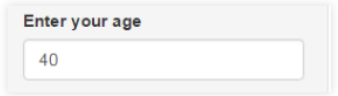
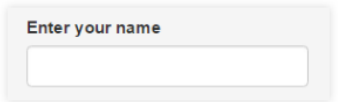
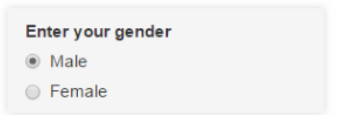
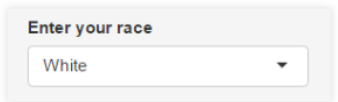

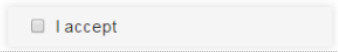
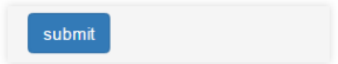


Outline

Part II: Form design

- [Input elements\[>\]](#)
- [Output elements\[>\]](#)
- [Layout\[>\]](#)
- [Conditional panels\[>\]](#)
- [Exercise\[>\]](#)

Input elements

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

Check [here](#) all input elements available in Shiny.

Drop-down list

- Use **selectInput**, **selectizeInput** functions.
- To select more than one item **multiple=TRUE**.
- When **multiple=TRUE**, drop-down list can be displayed in two formats:
 1. **Simple format**: items placed in a column (one below the other). Number of shown items (windows height) can be set by the **size** argument.
 2. **“selectize” format**: It allows to search items in case-sensitive typing, and add more options by the **options** argument using the **selectizeInput** function. For more info visit [this web](#)

```
library(compareGroups)
data(regicor)

ui <- fluidPage(

  selectInput("list1", "One option", names(regicor)),

  selectInput("list2", "Simple format", names(regicor),
    multiple=TRUE, selectize=FALSE),

  selectInput("list3", "Simple format", names(regicor),
    multiple=TRUE, size=ncol(regicor), selectize=FALSE),

  selectizeInput("list4", "Selectize format", names(regicor),
    multiple=TRUE,
    options=list(plugins=list('remove_button', 'drag_drop')))
)

server <- function(input, output){}

shinyApp(ui, server)
```

One option

id

Simple format

id
year
age
sex
smoker


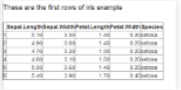
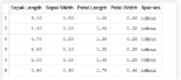

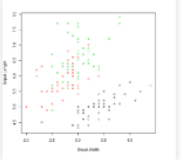

Simple format

id
year
age
sex
smoker
sbp
dbp
histtn
txhtn
chol
hdl
triglyc
ldl
histchol
txchol
height
weight
bmi
phyact
pcs
mcs
cv
tocv
death
todeath

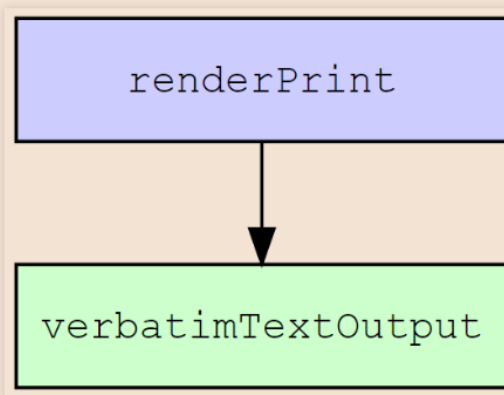
Selectize format

id × age × smoker ×

Output elements

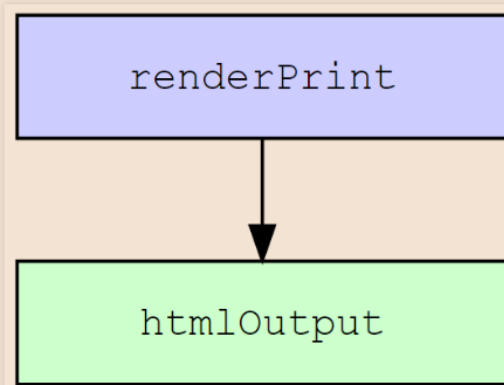
Type	Function	Arguments	Example
R-console like text	verbatimTextOutput	outputId	
HTML interpreted text	htmlOutput	outputId, inline	
“Regular” table	tableOutput	outputId	
Dynamic table	dataTableOutput	outputId	
Plots	plotOutput imageOutput	outputId, width, height, click, ...	
Dynamic plots	plotlyOutput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

R-console like text



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

HTML interpreted text



```
library(xtable)

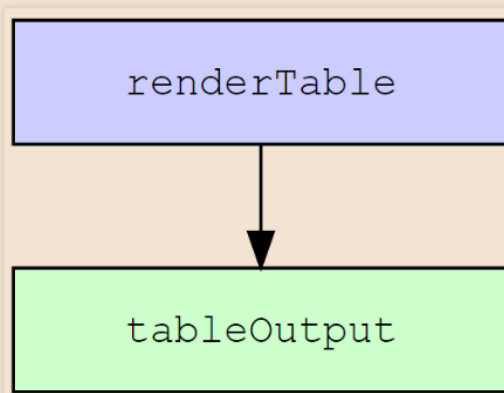
ui <- fluidPage(
  htmlOutput("result")
)

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

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

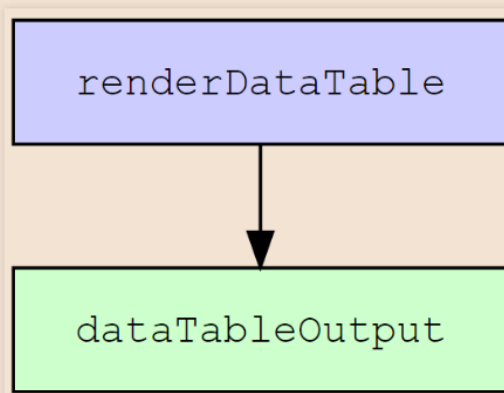
Exercise: Replace `htmlOutput` by `verbatimTextOutput`

"Regular" table



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

Dynamic table



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

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')
  )
)
```

Copy CSV Excel PDF Print

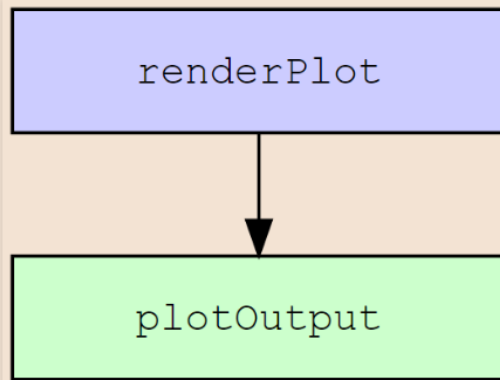
Search:

Sepal.Length ↕	Sepal.Width ↕	Petal.Length ↕	Petal.Width ↕	Species ↕
<input type="text" value="All"/>	<input type="text" value="All"/>	<input type="text" value="All"/>	<input type="text" value="All"/>	<input type="text" value="All"/>
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

Showing 1 to 10 of 150 entries

Previous 2 3 4 5 ... 15 Next

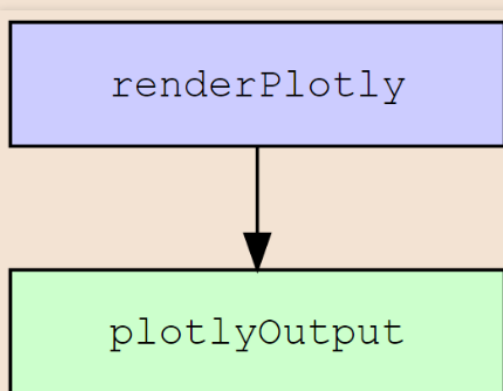
Plots



```
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?

Dynamic plots



```
library(plotly)

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

server <- function(input, output) {
  output$result<-renderPlotly({
    plot_ly(data=iris, x=~Sepal.Length,
    y=~Petal.Length)
  })
}

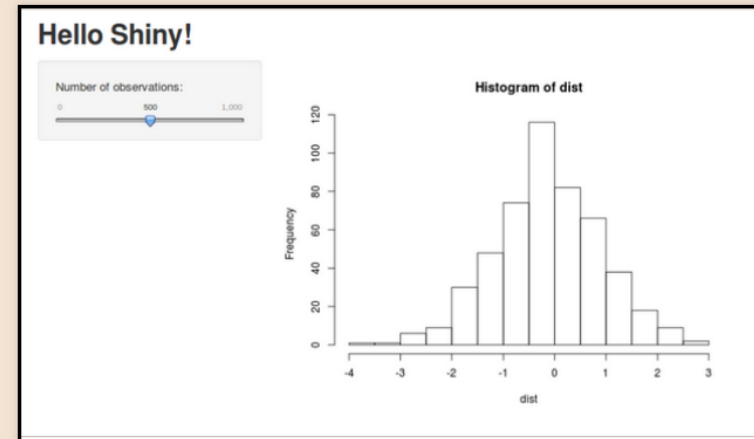
shinyApp(ui = ui, server = server)
```

- As for datatable, no need to plug into Shiny app

Layout

Left and right panels

```
fluidPage(  
  sidebarLayout(  
    sidebarPanel(...),  
    mainPanel(...)  
  )  
)
```



- This is the most common option.
- Equivalently, you can also use the function `bootstrapPage` instead of `fluidPage`.

Rows and columns specification

```
fluidPage(  
  fluidRow(  
    column(4, ...),  
    column(8, ...)  
  ),  
  fluidRow(  
    column(6, ...),  
    ...  
  ),  
  ...  
)
```

- This is the most flexible option.
- You can set the column width.
- Height of columns **can not** be set.
- The sum of columns widths must be 12.
- You can place as many columns as desired, even nested.
- Every row is created by **fluidRow** function.

```
ui <- shinyUI(fluidPage(  
  titlePanel("Grid example"),  
  ...  
  ...  
  actionButton("submit","")  
))  
server <- function(input, output) {}  
shinyApp(ui = ui, server = server)
```

Grid example

Enter your name

Enter your age

Enter your gender

☒ Male

☐ Female

This is a very long text. This is a very long text. This is a very long text. This is a very long text. This is a very long text.

Exercise: Complete the code to place the elements as in this example

General menu

```
navbarPage(  
  tabPanel(...),  
  tabPanel(...),  
)
```

- It is not as frequently used as the previous ones.
- The functionality is the same as for tabs which will be explain below.
- Usefull for “big” applications than can be split in “sub-applications”, one for each general menu tab.

General menu Students Teachers Subjects

Name and surnames

Age

Gender

☒ boy

☐ girl

General menu Students Teachers Subjects

Name and surnames

Number of subjects

Department

Maths ▼

Tabs

```
ui <- fluidPage(
  tabsetPanel(id = "menu",
    tabPanel("Table",
      ...
    ),
    tabPanel("Summary",
      ...
    )
  )
)
```

Tabla	Resumen				
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa

Tabla	Resumen				
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa	:50
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	versicolor	:50
Median :5.800	Median :3.000	Median :4.350	Median :1.300	virginica	:50
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		

- Elements are arranged one **behind** the other.
- Use **tabsetPanel** function to create a set of tabs.
- Use **tabPanel** function to create each single tab.

Merging several tabs (drop-down menu)

- Use **navbar** function with the tabs as its arguments.
- It is used inside **tabsetPanel** to create tabs.
- Although one tab does not contain a drop-down menu, **tabPanel** is used.
- You can specify the appearance with “type” argument of **tabsetPanel** function.

```
ui <- fluidPage(  
  tabsetPanel(id="menu", type="pills",  
    tabPanel("Tab 1",  
      ...  
    ),  
    navbarMenu("Drop-down",  
      tabPanel("Option 1",  
        ...  
      ),  
      tabPanel("Option 2",  
        ...  
      )  
    )  
  )  
)
```

Summary		Table	
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100
1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300
Median :5.800	Median :3.000	Median :4.350	Median :1.300
Mean :5.843	Mean :3.057	Mean :13.758	Mean :1.199
3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:15.100	3rd Qu.:1.800
Max. :7.900	Max. :4.400	Max. :16.500	Max. :2.500
Species			
setosa	:50		
versicolor	:50		
virginica	:50		

Summary	Table			
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

Conditional Panels

Conditional Panels

- Conditional panels are used when some elements must be shown or not depending on some other input elements values.
- **conditionalPanel** function.
 - First argument: character. Logic expression written in “**javascript**” language.
 - Second argument: form elements that will be appear when the logic expression is TRUE.

```
conditionalPanel(  
  condition = "input.element==1",  
  ...  
)
```

Example 1

```
ui <- fluidPage(
  radioButtons("type",
    "What do you want to show?",
    c("Table"=1, "Summary"=2)),
  conditionalPanel(
    condition = "input.type==1",
    tableOutput("result1")
  ),
  conditionalPanel(
    condition = "input.type==2",
    verbatimTextOutput("result2")
  )
)

server <- function(input, output) {
  output$result1 <- renderTable(
    head(iris)
  )
  output$result2 <- renderPrint(
    summary(iris)
  )
}

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

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

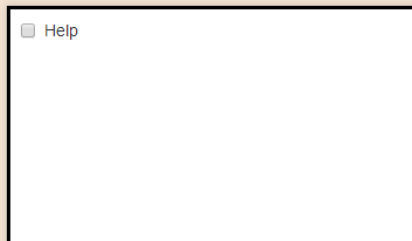
☒ Summary

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	versicolor
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	

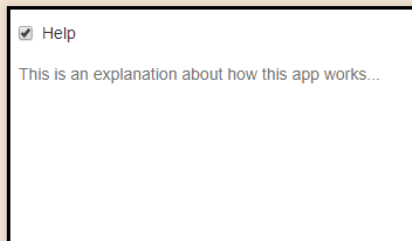
Exercise: Reproduce this app
using **tabsetPanel**

Example 2

```
ui <- fluidPage(  
  checkboxInput("help", "Help"),  
  
  conditionalPanel(  
    condition = "input.help",  
    helpText("This is an explanation  
              about how this app works...")  
  )  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```



☐ Help



☒ Help
This is an explanation about how this app works...

This text appears and disappears
clicking on the check box.

Example 3

```
library(shiny)

ui <- fluidPage(

  actionButton("toggle", "Toggle"),

  conditionalPanel(
    condition = "input.toggle%2==0",
    helpText("This is an explanation
              about how this app works...")
  )
)

server <- function(input, output) {}

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

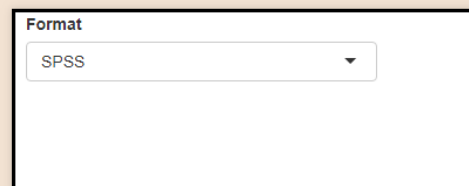
Toggle

Toggle

This is an explanation about how this app works...

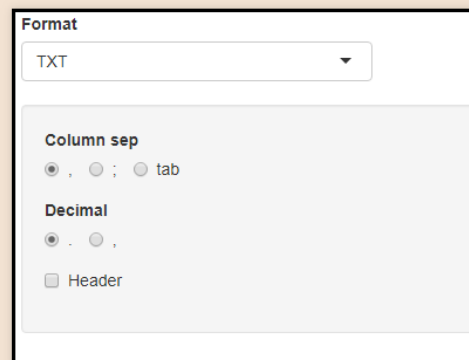
Example 4

```
ui <- fluidPage(  
  selectInput("format", "Format",  
    c("SPSS"=1, "EXCEL"=2, "TXT"=3, "R"=4)),  
  conditionalPanel(  
    condition = "input.format==3",  
    wellPanel(  
      radioButtons("sep", "Column sep",  
        c(",", ";", "tab"), inline = TRUE),  
      radioButtons("dec", "Decimal",  
        c(".", ","), inline = TRUE),  
      checkboxInput("header", "Header")  
    )  
  )  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```



Format

SPSS



Format

TXT

Column sep

☒ , ☐ ; ☐ tab

Decimal

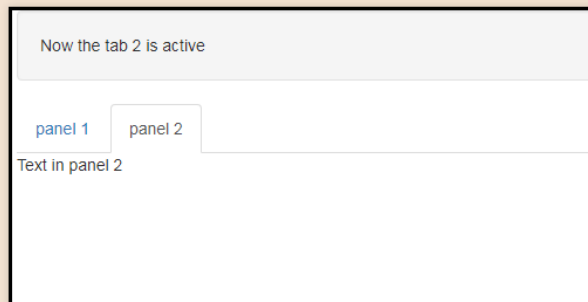
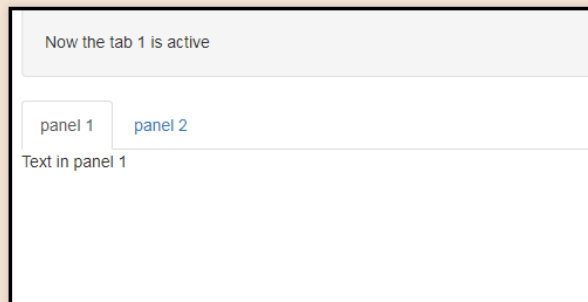
☒ . ☐ ,

☐ Header

Note the **wellPanel** to create a frame around the elements, and the **inline** argument of **radioButtons** function to place the items in horizontal.

Example 5

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      conditionalPanel(  
        condition = "input.menu=='panel 1'",  
        "Now the tab 1 is active"  
      ),  
      conditionalPanel(  
        condition = "input.menu=='panel 2'",  
        "Now the tab 2 is active"  
      )  
    ),  
    mainPanel(  
      tabsetPanel(id = "menu",  
        tabPanel("panel 1",  
          "Text in panel 1"  
        ),  
        tabPanel("panel 2",  
          "Text in panel 2"  
        )  
      )  
    )  
  )  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```



The result is different depending on the active tab.

Exercise

Complete the code in the UI part to place the elements and tabs in the following form:

On the **left hand** of the form it must be a panel where:

- the user can choose one of the following distributions
 - exponential
 - normal
 - binomial
- Depending on the distribution selected, proper parameters must appear so that the user can enter their values:

On the **right hand** of the form there must be two tabs:

- First tab containing a histogram with 1,000 data randomly generated under the selected distribution with the specified parameters.
- Second tab containing a summary of generated data.

Distribution exercise

Choose distr

Binomial

Trials

10

Prob

0

0.5

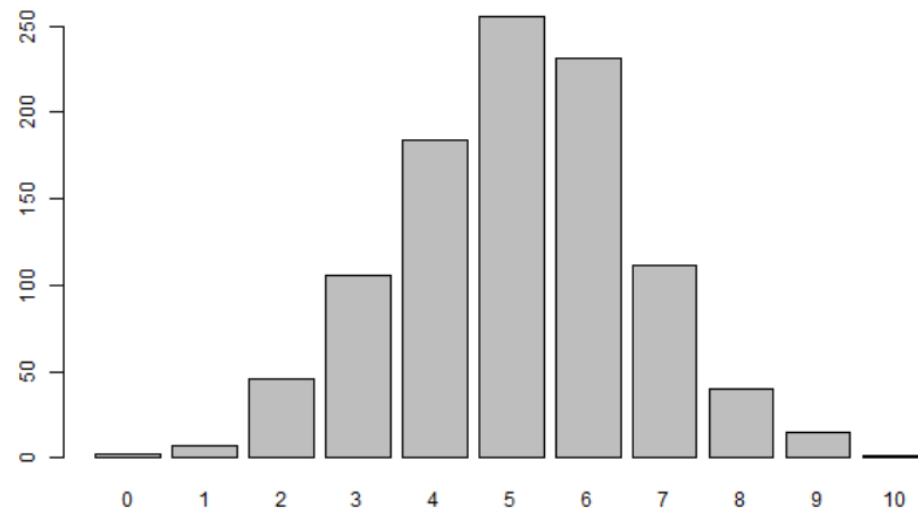
1

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Plot

Summary

Distr: Binomial



```
ui <- fluidPage(  
  titlePanel("Distribution exercise"),  
  sidebarLayout(  
    sidebarPanel(  
      ## .. to be completed  
    ),  
    mainPanel(  
      ## .. to be completed  
    )  
  )  
)
```

```
server <- function(input, output) {  
  output$summary <- renderPrint({  
    if (input$distr=="Normal")  
      data <- rnorm(1000, input$mu, input$sd)  
    if (input$distr=="Exponential")  
      data <- rexp(1000, input$lambda)  
    if (input$distr=="Binomial")  
      data <- rbinom(1000, input$n, input$p)  
    summary(data)  
  })  
  output$plot <- renderPlot({  
    if (input$distr=="Normal")  
      data <- rnorm(1000, input$mu, input$sd)  
    if (input$distr=="Exponential")  
      data <- rexp(1000, input$lambda)  
    if (input$distr=="Binomial")  
      data <- rbinom(1000, input$n, input$p)  
    if (input$distr=="Binomial"){  
      datafact <- factor(data, levels=0:input$n)  
      barplot(table(datafact))  
    } else {  
      hist(data, main="")  
    }  
    title(paste("Distr: ", input$distr))  
  })  
}
```