Tutorial: development of an online risk calculator platform

Xinge Ji, Michael W. Kattan

Department of Quantitative Health Sciences, Lerner Research Institute, Cleveland Clinic, Cleveland, Ohio, USA

Correspondence to: Michael W. Kattan, PhD. Department of Quantitative Health Sciences, Lerner Research Institute, Cleveland Clinic, 9500 Euclid Avenue/JJN3-01, Cleveland, Ohio 44195, USA. Email: kattanm@ccf.org.

Abstract: Risk calculators are online tools developed for use by physicians in clinical settings to predict the risk of a clinical event, and as an aid in personalizing medical decision-making. Cleveland Clinic prediction models are listed at http://rcalc.ccf.org. We illustrate how we used R to create a risk calculator, and demonstrate the ease of using R, RStudio, and a Shiny package.

Keywords: Online risk calculator; shiny server; platform

Submitted Nov 21, 2017. Accepted for publication Nov 24, 2017.
doi: 10.21037/atm.2017.11.37
View this article at: http://dx.doi.org/10.21037/atm.2017.11.37

Introduction

Risk calculators are online tools physicians used to predict the risk of a clinical event. Our Cleveland Clinic Prediction models are listed at http://rcalc.ccf.org.

If R (1) is new to you, go to the learning resources at https://www.rstudio.com/training. We will build a risk calculator as a Shiny application. For complete tutorials about Shiny, see https://shiny.rstudio.com/. Basic HTML and cascading style sheets (CSS) knowledge is required in section 2.5; however, that section may be skipped.

You'll need to have R and RStudio (2) installed on your desktop. R will compile and run on a wide variety of UNIX platforms, Windows, and MacOS. In addition, you will need to install the Shiny package (3). Open an R session, connect to the internet, and run: install.packages("shiny").

Risk calculator

The Shiny app contains two scripts called ui.R and server.R. The scripts live in a directory, e.g., app/, and the app can be run with `runApp("app")`. Below we work through an example of building an "Ideal Weight Calculator".

Create a new directory named app in your working directory. Copy and paste the following scripts into the directory (the bullets are the filenames, followed by the file content).

❖ ui.R

```
library(shiny)

fluidPage(
# App title
  titlePanel("Ideal Weight Calculator"),

  # Sidebar layout with input and output definitions
  sidebarLayout(
    # Sidebar panel for inputs
    sidebarPanel(
      # Input: text input for Height
      textInput("Height", "Height (cm)"),
      # Input: select list input for Gender
          selectInput("Gender", "Gender", choices =
c("Male", "Female"))
    ),

    # Main panel for displaying outputs
    mainPanel(
      # Output: Table
      tableOutput("result")
    )
  )
)
```

❖ server.R

```
# Define server logic required to create a table
shinyServer(function(input, output){
  # It is a function that creates a dataframe called data for the inputs
  # The function is "reactive" and therefore should be automatically
  # re-executed when inputs change
  inputdata <- reactive({
    data <- data.frame(
      MyHeight = as.numeric(input$Height),
      MyGender = input$Gender
    )
    data
  })

  # Table to display the ideal body weight
  output$result <- renderTable({
    # Executes the "inputdata" function to save the dataframe as "data"
    data = inputdata()
    # Determine the ideal weight by "MyGender" and "MyHeight"
    if (data$MyGender == "Male") {
      idealWeight = 50 + 0.9 * (data$MyHeight - 152)
    } else {
      idealWeight = 45.5 + 0.9 * (data$MyHeight - 152)
    }
    # create a dataframe for output
    resultTable = data.frame(
      Result = "Your ideal weight (kg) is",
      Weight = idealWeight
    )
    resultTable
  })
})
```

Your directory should look like this:

~/app/ui.R

~/app/server.R

Open either the ui.R script or the server.R script in your RStudio editor, then launch the app by clicking the "Run App" button (see *Figure 1*) or use the keyboard: Ctrl + Shift + Enter (Cmd + Shift + Enter on the MacOS) (see *Figure 1*).

The new app should match *Figure 2* below. You can input a number into the text box for patient height, and select male or female in the gender box and see the change in the Result box (see *Figure 2*).

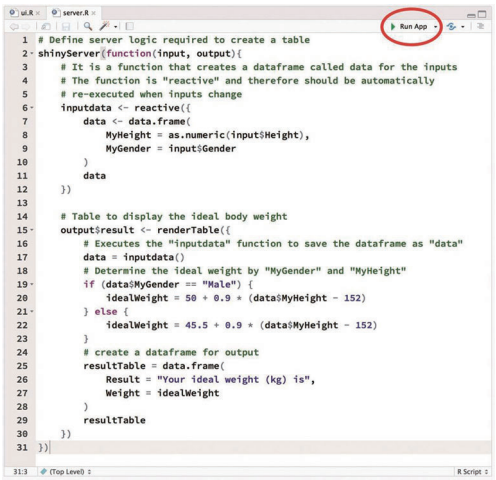

Figure 1 An example of launching Shiny app in RStudio.

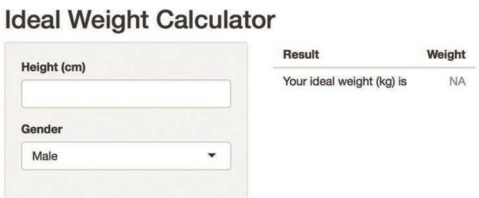

Figure 2 The ideal weight calculator Shiny App.

Using control widgets

A control widget is a web element with which users interact. Shiny widgets collect a value from the user. When the user changes the widget, the value will change as well. To add a widget to the app, place a widget function in the sidebarPanel or mainPanel in the ui.R. Each widget function requires several parameters. The first two parameters for each widget are the following:

- ❖ Name: you can use the name to access the widget's value;
- ❖ Label: the label will appear in your app.

In the first example, we have seen the selectInput widget and textInput widget.

SelectInput widget

In the selectInput widget, the name is Gender and the label

is also Gender. There is another parameter called choices which gives a vector of available values for the widget. In our example, the choices for gender are Male and Female:

```
selectInput("Gender", "Gender", choices = c("Male", "Female")).
```

TextInput widget

In the textInput widget, the name is Height and the label is Height (cm):

```
textInput("Height", "Height (cm)").
```

There are two additional optional parameters for textInput:

- Value: initial value;
- Placeholder: a character string giving the user a hint as to what can be entered into the control.

For example, we can set 170 as the default value of Height, and add a placeholder 150 - 200 to indicate an appropriate range for the calculator:

```
textInput("Height", "Height (cm)", value = "170", placeholder = "150 - 220").
```

In order to make sure the user input is valid, it makes sense to validate the input value in the reactive function of server.R. The following code validates that the input of Height is a numerical value and is within 150 and 220. If not validated a warning message "Please input a valid value for Height" will show in the output.

```
validate(need(!is.na(as.numeric(input$Height)) &
              as.numeric(input$Height)>=150 &
              as.numeric(input$Height)<=220,
            "Please input a valid value for Height"))
```

The complete scripts are:

- ui.R

```
library(shiny)

fluidPage(
  # App title
  titlePanel("Ideal Weight Calculator"),

  # Sidebar layout with input and output definitions
  sidebarLayout(
    # Sidebar panel for inputs
    sidebarPanel(
      # Input: text input for Height
      textInput("Height", "Height (cm)",
                value = 170, placeholder = "150 - 220"),
      # Input: select list input for Gender
      selectInput("Gender", "Gender",
                  choices = c("Male", "Female"))),

    # Main panel for displaying outputs
    mainPanel(
      # Output: Table
      tableOutput("result")
    )
  )
)
```

- server.R

```
# Define server logic required to create a table
shinyServer(function(input, output){
  # It is a function that creates a dataframe called data for the inputs
  # The function is "reactive" and therefore should be automatically
  # re-executed when inputs change
  inputdata <- reactive({
    validate(need(!is.na(as.numeric(input$Height)) &
                  as.numeric(input$Height)>=150 &
                  as.numeric(input$Height)<=220,
                "Please input a valid value for Height"))

    data <- data.frame(
      MyHeight = as.numeric(input$Height),
      MyGender = input$Gender
    )
    data
  })

  # Table to display the ideal body weight
  output$result <- renderTable({
    # Executes the "inputdata" function to save the data-frame as "data"
    data = inputdata()
    # Determine the ideal weight by "MyGender" and "MyHeight"
    if (data$MyGender == "Male") {
      idealWeight = 50 + 0.9 * (data$MyHeight - 152)
```

```
    } else {
      idealWeight = 45.5 + 0.9 * (data$MyHeight - 152)
    }
    # create a dataframe for output
    resultTable = data.frame(
      Result = "Your ideal weight (kg) is",
      Weight = idealWeight
    )
    resultTable
  })
})
```

NumericInput widget

NumericInput is used as an input control for entry of numeric values. The parameter value is required, and min and max are optional. NumericInput and textInput are interchangeable when it comes to a widget for numeric values.

SliderInput widget

Another option is using sliderInput instead of textInput for numeric input, the parameters min, max and value are required:

sliderInput("Height", "Height (cm)", min = 150, max = 220, value = 170).

This widget is not as flexible as textInput. For example, if the predictor is optional, or if the value of the predictor does not have a minimum or maximum, the variable cannot fit in the sliderInput widget.

Predictive expression

The code of the ideal weight calculation can be reorganized as a separate expression which would make it more readable and maintainable. The reason for this is that it splits the long code sequences into short pieces and people can easily find the code for prediction model. If the prediction model needs updating in the future, changes in the expression won't break the code sequences for the server.

❖ server.R

```
# Expression to predict the ideal weight
ideal.weight <- expression({
  if (data$MyGender == "Male") {
    idealWeight = 50 + 0.9 * (data$MyHeight - 152)
  } else {
    idealWeight = 45.5 + 0.9 * (data$MyHeight - 152)
  }
  idealWeight
})

# Define server logic required to create a table
shinyServer(function(input, output){
  # It is a function that creates a dataframe called data for the inputs
  # The function is "reactive" and therefore should be automatically
  # re-executed when inputs change
  inputdata <- reactive({
    validate(need(!is.na(as.numeric(input$Height)) &
                  as.numeric(input$Height)>=150 &
                  as.numeric(input$Height)<=220,
                "Please input a valid value for Height"))

    data <- data.frame(
      MyHeight = as.numeric(input$Height),
      MyGender = input$Gender
    )
    data
  })

  # Table to display the ideal body weight
  output$result <- renderTable({
    # Executes the "inputdata" function to save the data-frame as "data"
    data = inputdata()
    # Evaluate the "ideal.weight" expression with values from "data"
    idealWeight = eval(ideal.weight, data)
    # create a dataframe for output
    resultTable = data.frame(
      Result = "Your ideal weight (kg) is",
      Weight = idealWeight
    )
    resultTable
  })
})
```

Output

We use the renderTable in server.R and tableOutput in ui.R to display the result as a table in our example. RenderTable is a widget used to render static tables in a Shiny app, and tableOutput renders the renderTable within the app page.

The Shiny package also supports rendering other types of reactive output variables such as text and plot. Please check out the reference of the package for usage.

Advanced topics

These require some knowledge of HTML and CSS. However, these modifications can enhance the risk calculator substantially by improving quality interaction between a user and all information a user should know about the calculator.

WellPanel

A well panel has a slightly inset border and grey background. Let's put the output table inside a well panel and the reference for the calculator in another well panel. It visually splits the two components and improves the user experience.

❖ ui.R

```
library(shiny)

fluidPage(
  # App title
  titlePanel("Ideal Weight Calculator"),

  # Sidebar layout with input and output definitions
  sidebarLayout(
    # Sidebar panel for inputs
    sidebarPanel(
      # Input: text input for Height
      textInput("Height", "Height (cm)",
                value = 170, placeholder = "150 - 220"),
      # Input: select list input for Gender
      selectInput("Gender", "Gender",
                  choices = c("Male", "Female"))),

    # Main panel for displaying outputs
    mainPanel(
      # well panel for output
      wellPanel(
        # Output: Table
        tableOutput("result")),
      # well panel for reference
      wellPanel(
        # link is simplified by Google URL shorter
          p(a("Devine formula", href="https://goo.gl/
brjjjZ")),
        p("Men: Ideal Body Weight (kg) =
          50 kilograms + 0.9 kilograms × (height (cm) −
152)"),
        p("Women: Ideal Body Weight (kg) =
          45.5 kilograms + 0.9 kilograms × (height (cm)
− 152)")
      )
    )
  )
)
```

Hide and show

Sometimes we want to hide (or show) some components in the user interface. We can use functions from the shinyjs (https://github.com/daattali/shinyjs) package to accomplish this. For example, we might want to restrict to users who are at least 18 years old to be able to access the calculator. The following code implements this restriction.

❖ ui.R

```
library(shiny)
library(shinyjs)

fluidPage(
  useShinyjs(),
  # App title
  titlePanel("Ideal Weight Calculator"),
  # Check user"s age
  radioButtons("age", "Are you at least 18 years of age",
               choices = c("No", "Yes"), selected = "No"),
  hidden(
    div(
      id= "age18",
      # Sidebar layout with input and output definitions
      sidebarLayout(
```

```
        # Sidebar panel for inputs
        sidebarPanel(
          # Input: text input for Height
          textInput("Height", "Height (cm)",
                    value = 170, placeholder = "150 - 220"),
          # Input: select list input for Gender
          selectInput("Gender", "Gender",
                      choices = c("Male", "Female"))),

        # Main panel for displaying outputs
        mainPanel(
          # well panel for output
          wellPanel(
            # Output: Table
            tableOutput("result")),
          # well panel for reference
          wellPanel(
            # link is simplified by Google URL shorter
              p(a("Devine formula",href="https://goo.
gl/brjjjZ")),
            p("Men: Ideal Body Weight (kg) =
               50 kilograms + 0.9 kilograms × (height
(cm) − 152)"),
            p("Women: Ideal Body Weight (kg) =
              45.5 kilograms + 0.9 kilograms × (height
(cm) − 152)")
          )
        )
      )
    )
)
```

❖ server.R

```
# Function to predict the ideal weight
ideal.weight <- expression({
  if (data$MyGender == "Male") {
    idealWeight = 50 + 0.9 * (data$MyHeight - 152)
  } else {
    idealWeight = 45.5 + 0.9 * (data$MyHeight - 152)
  }
  idealWeight
})

# Define server logic required to create a table
shinyServer(function(input, output){
  # Show/hide the content in the "age18" div
  observe({
    if (input$age == "Yes") {
      show(id = "age18", anim = TRUE)
    } else {
      hide(id = "age18", anim = TRUE)
    }
  })

   # It is a function that creates a dataframe called data
for the inputs
   # The function is "reactive" and therefore should be
automatically
  # re-executed when inputs change
  inputdata <- reactive({
    validate(need(!is.na(as.numeric(input$Height)) &
                  as.numeric(input$Height)>=150 &
                  as.numeric(input$Height)<=220,
                "Please input a valid value for Height"))

    data <- data.frame(
      MyHeight = as.numeric(input$Height),
      MyGender = input$Gender
    )
    data
  })

  # Table to display the ideal body weight
  output$result <- renderTable({
     # Executes the "inputdata" function to save the data-
frame as "data"
    data = inputdata()
     # Determine the ideal weight by "MyGender" and
"MyHeight"
    idealWeight = eval(ideal.weight, data)
    # create a dataframe for output
    resultTable = data.frame(
      Result = "Your ideal weight (kg) is",
      Weight = idealWeight
    )
```

```
    resultTable
  })
})
```

Themes

Appearance of the Shiny app can be altered with CSS, a widely used language for describing the visual style of web pages. For pre-built themes, please check out the shinythemes package (https://rstudio.github.io/shinythemes/) and the shinydashboard package (https://rstudio.github.io/shinydashboard/).

Risk calculator deployment

There are several options for putting the calculator up on the web:

Deploy to the Shinyapps.io (http://www.shinyapps.io/); this is easy to use. No hardware or installation is required. Free and paid options are available.

Deploy with the open source Shiny Server (https://www.rstudio.com/products/shiny/shiny-server/);

Deploy the Shiny apps and interactive documents on-premises with open source Shiny Server, like what is done at http://rcalc.ccf.org/. The official configuration reference can be found at http://docs.rstudio.com/shiny-server/. We deploy our server on the Amazon Web Services (AWS) framework. All files for Shiny apps are examined to make sure that no data will be saved or modified on the server before moving to the Shiny Server. We provide only published models on the main index page and share unpublished models with direct links. We enable Google Analytics to collect user behaviors.

Collaborate with us (rcalcsupport@ccf.org).

Cite this article as: Ji X, Kattan MW. Tutorial: development of an online risk calculator platform. Ann Transl Med 2018;6(3):46. doi: 10.21037/atm.2017.11.37

Acknowledgements

The authors would like to thank Stephanie Kocian for her editing of the manuscript.

Footnote

Conflicts of Interest: The authors have no conflicts of interest to declare.

References

1. R Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing, 2017. Available online: https://www.R-project.org/
2. RStudio Team. RStudio: Integrated Development Environment for R. Boston, MA: RStudio, Inc., 2016. Available online: http://www.rstudio.com/
3. Chang W, Cheng J, Allaire JJ, et al. Shiny: Web Application Framework for R. 2017. Available online: https://CRAN.R-project.org/package=shiny

