Let R Shiny do the storytelling job for you and add fantastic interactive interfaces for your R analysis.

Why should you care about learning shiny? Follow me in this not so improbable scenario and you’ll see why.

Let’s say you have an R function my\_hist that reads in a data set x and takes a parameter

bins to return a histogram.

#' Function to perform my awesome analysis #' Returns a plot of the analyzed data

#' @param x data frame containing the data to analyze

#' @param bins integer parameter indicating the number of bins for the histogram. Default value is 30.

#' @importFrom graphics hist my\_hist <- function(x, bins = 30){

# draw the histogram with the specified number of bins hist(

x, breaks = bins, main = "Histogram",

col = "darkgray", border = "white"

)

}

If you wish your colleague John to look at the results of your analysis, you should provide a document with your plot, and the references to the data and the parameters used. Now let’s assume he wants to see the changes in the results using 20 bins instead of 30. John would have to reach out to you, you would run the analysis again, send him the new results and repeat this in a loop until John is happy with the results / gets an understanding of the data from a business perspective. Hopefully at this point you both still know which results corresponds to which data and set of parameters, otherwise this is bound to become a mess.

Would it not be much easier if John could just take your code and run all his scenarios on his own? Yes! However, he does not know R, doesn’t have it installed or doesn’t have your same set up… all potential sources of errors, misalignment and chaos.

With R shiny you can easily solve this problem and provide an interactive user interface to your function.

#' Shiny App User Interface #' @import shiny

ui <- function(){ fluidPage(

# Application title

titlePanel("Histogram Old Faithful Data Set"), # Layout with a sidebar and a main panel sidebarLayout(

# Panel with parameters controls sidebarPanel(

sliderInput(

inputId = "bins", label = "Number of bins:", min = 1, max = 50, value = 30

)

),

# Panel showing the results mainPanel(

plotOutput(outputId = "my\_plot")

)

)

)

}

#' Shiny App server logic

#' @param id,input,output,session Internal parameters for {shiny}. #' @import shiny

server <- function(input, output, session) { x <- datasets::faithful[, "waiting"]

# Reactive parameter for the bins

bins <- reactive(seq(min(x()), max(x()), length.out = input$bins + 1))

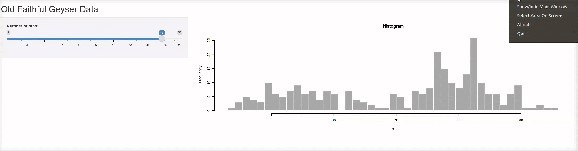
# Construct reactive plot output$my\_plot <- renderPlot({

my\_hist(x(), bins())

})

}

shinyApp(ui,server)



With few lines of code you can build a dynamic web application John can easily interact with. All in R and without having to become a web developer!

This is a minimal example of what you can do with shiny. You can easily go from a quick prototype for internal use, to a more complex and professional application that can be used in a productive environment. With a few lines of code you can, for example, build downloadable reports showing the results of your analysis and all the parameters needed for full reproducibility; you can add information and documentation for the user directly in the interface; you can also extend the application with additional plots and features, or you can reuse the same modular analysis in a different context. In a few words, the possibilities that R shiny opens up are only limited by the data scientist imagination.

If you do not yet know how to build a shiny app, **have a look at Mirai’s upcoming workshop “Build your first Shiny App” and learn with us how to develop a shiny app from scratch**. It is the first of a series of workshops focusing on shiny.