Building Interactive Visualizations with Shiny

Marcus A. Nunes

1 Introduction

Shiny (Chang et al. 2023) is an R (R Core Team 2023) package and framework whose goal is to simplify the creation of interactive web applications for R users. Interactive web applications demand other programming languages, such as HTML, CSS, and Javascript (Wickham 2021) to be built. Usually, R users do not have this knowledge because their focus is generally on Statistics and Probability problems. In this sense, R is more like a tool for them to achieve the results they want regarding data analysis than an end in itself. Although many R users can write code in other languages, such as C/C++, FORTRAN, SAS, or Python, they are all focused on the scientific use of these languages.

One can employ user interfaces without knowing how to create them using Shiny. Pre-built interfaces can be selected and adapted according to the user's needs. This way, the programmers do not need to build the app interface by themselves. They just write the R code of their statistical application and the link between it and the web page is taken care of automatically. In this sense, users do not need to learn another programming language before building their own applications.

But what kind of applications can be built using Shiny? Essentially, any R code that receives an input and gives an output can be transformed into a Shiny app. Some use cases are:

- Companies that need to generate personalized reports for different managers can create a Shiny app with parameters that can be changed according to their needs instead of reports with hundreds (or even thousands) of PDF pages (Sievert 2020)

M. A. Nunes (✉)
UFRN, Natal, Brazil
e-mail: marcus.nunes@ufrn.br

O. O. Awe, E. A. Vance (eds.), *Practical Statistical Learning and Data Science Methods*, STEAM-H: Science, Technology, Engineering, Agriculture, Mathematics & Health, https://doi.org/10.1007/978-3-031-72215-8_13

- Professors that want to create interactive visualizations to facilitate the understanding of statistical or mathematical concepts to their students (Jak et al. 2021)
- Simulations of complex analytic models where parameters can be changed and the user can see possible outcomes without the need to run experiments (Wojciechowski et al. 2015)
- Create a simplified interface for specific data analysis where the users can upload their datasets and get an automatic report with tables, plots, and statistical analysis (Ekiz et al. 2020).

All these Shiny apps can be hosted on the web, making them available to anyone worldwide. However, if the application is meant to be used by a few people, it can be hosted on an intranet or a password-protected website.

To achieve these results, Shiny has some features that may be unique for some R users. Some of these features are:

1. Reactive programming: This programming paradigm enables real-time reactions to data and input changes. As soon as the user changes some options in the application, the new results are computed and shown. It is a great resource to build interactive applications.
2. User interface: Due to its nature, Shiny apps can be divided into two parts, namely, *user interface* and *server*. Instead of editing code directly, Shiny offers boxes for inputs, sliders, checkboxes, dropdown menus, and other components that can be used to help the user interact with the app. The programmer can customize these components to fit the app's appearance.
3. Server-side processing: The server-side component is responsible for the app's logic and calculations. All tasks like data import and cleaning, model fitting, plots, and everything an R program can do is run on the server. If the app is hosted on the Internet, the computer power is defined by the app host and not the user's personal computer, tablet, or cell phone.
4. Web design: Shiny has many templates that help the programmer create web pages with HTML, CSS, and JavaScript integrated. These templates can be customized according to the programmer's taste, but it is also possible for each person to develop their design.
5. Software deployment: Applications built with Shiny can be deployed on the Internet to be shared with people around the world. The apps can be run locally, but the real Shiny power comes with app web hosting. Besides the easy access, bugs and updates can be easily applied if the app is hosted on an external server.

Therefore, the Shiny R package is an excellent option for statisticians, data scientists, and educators who want or need to build interactive web applications using R. It is practical and user-friendly interfaces can be used to explore and present data, transforming it into a valuable option for data visualization, communication, and education. Moreover, building applications with Shiny only requires prior experience with the R programming language.

2 Using Shiny

2.1 Getting Shiny

Since this text is focused on intermediate to advanced R users, it is essential to note that previous experience with the language is advisable. Generally, it is not recommended that we learn Shiny and R together.

As with most R packages, Shiny can be downloaded from CRAN. With R already installed, one needs to run the command

```
install.packages("shiny", dependencies = TRUE)
```

In the R prompt, install the package. This method works for all R versions and operating systems. The computer only needs to be connected to the Internet.

Another method is to install Shiny using an IDE (Integrated Development Environment). RStudio (https://posit.co/download/rstudio-desktop/) is a well-known IDE for R, and it provides a menu for package installation. One just needs to click on Tools > Install Packages . . . menu and RStudio install Shiny and all its dependencies in your machine.

2.2 Structure of a Shiny Application

A Shiny application has two mandatory parts: *user interface* and *server* (Wickham 2021). The user interface is how the Shiny programmer defines how other people interact with the Shiny app. It defines how the app looks. The server is where the app processing is made. It describes how the app works.

The user interface can be used to enter app parameters. There are built-in functions to define different types of data input, like sliders, dropdown menus, radio buttons, text boxes, and so on. We will see how to use some of them in the application we build in the next section.

Regular R commands are inside the server. Every input given in the user interface is passed to the server and executed by it. If one needs to filter, transform, or plot data, among many more instructions, the server takes care of it with a twist. Since Shiny operates on reactive programming, as described in the previous section, the user interface can be updated in real-time after the server runs its commands.

2.2.1 User Interface

The programmer must create the user interface. This is where the look and feel of the app is defined. In other words, it dictates the app behavior for the user. It includes elements for the user to choose from and interact with the data or analysis the app provides.

A Shiny interface has a layout that organizes different parts of the app. The inputs and outputs go in it, usually with some text explaining what the app is about. It can be straightforward, with the content organized in rows and columns, or complex, with tabs and sidebars.

The inputs in the user interface are the elements that are used for interaction with the app. They can be text inputs, radio buttons, sliders, checkboxes, file inputs, etc. They are chosen according to the app's goals. If the user needs to select only one of many options, radio buttons are used, but if many options can be chosen, then checkboxes are considered.

Like the inputs, the outputs depend on the app's goals. They show the output generated by the app, such as plots, tables, or text. Since Shiny uses reactive programming, the outputs are updated once the inputs are changed.

Text elements can be added to the app to make it easier. They include headers, footers, descriptions, or any other textual content that provides instructions or context to the user. Usually, a welcome text explaining what the app is about helps the user understand how to interact with it.

In short, the user interface has everything the user needs to know about the app, from data entry and parameters to displaying plots and results. It can provide feedback to users, such as displaying messages or notifications to confirm actions, provide instructions, or alert users about errors or warnings.

2.2.2 Server

While the user interface is the front end of the Shiny app, the server is the back end. Every R function the app needs to evaluate is performed on the server. Notice that the code is evaluated as reactive expressions, and therefore, they are automatically re-executed whenever their dependencies, such as user inputs, change. It contains the instructions for the app's reactivity, meaning it specifies how the app should respond when users interact with it.

The server listens for changes in user input and updates the outputs accordingly. When a user interacts with control parameters on the UI, such as sliders or buttons, these inputs are passed to the server function. The server function then uses these values to perform computations, manipulate data, and generate the desired outputs like plots, tables, or text.

To create outputs, the server function employs a set of render functions that correspond to the type of output needed. For instance, `renderPlot` for plots, `renderDataTable` for tables, and `renderText` for text outputs. Each output is built using these render functions and assigned to the output object, a list-like structure containing all the code necessary to update the R objects in the app. Shiny automatically manages the process of running these instructions when the app is launched and re-running them whenever an update is needed due to a change in user input. All the calculations performed by the server are made using regular R code mixed with Shiny-specific commands.

The server function typically takes two primary arguments: input and output. Sometimes, it may also take a third argument, session, which provides access to information and functionality related to the user's session. The server function is called when a client (web browser) first loads the Shiny application's page, and any return value from this function is ignored. The server function's logic involves mapping user inputs to outputs, ensuring the app behaves as intended.

In the next section, a Shiny app will be created from scratch. We will see how the user interface and the server are joined to build an app.

3 Application

The creation of a Shiny app will be described in this section. The goal at the end of the chapter is to have a way to visualize economic data from the World Bank (2024). The code for the app is available at https://github.com/mnunes/world_bank_shiny. You should download the code and follow this section to see how a Shiny app works in practice.

The entire code for a shiny app can be stored in a single .R file. I prefer to have at least three files, namely, `global.R`, `ui.R`, and `server.R`, as one can see in the linked GitHub repository. I use `global.R` to load all packages and data my application uses, while `ui.R` and `server.R` have the codes for the user interface and server commands, respectively. The code becomes more accessible to audit and debug this way.

If I am working in a more sophisticated app with many specialized functions or different data sources, I usually create folders called `functions` and `data` to organize them. For this app, I chose to create a `data` folder to store the .csv file that will be used to develop the app. This .csv file was obtained from World Bank, and it has the following columns:

- country: name of the country
- region: region of the world where the country is located
- year: year of the statistic
- population: number of people living in the country
- life_expectancy: life expectancy at birth
- gdp: Gross Domestic Product in US$
- gdp_per_capita: GDP nominal per capita
- fertility: number of children per woman
- poverty: percent of the country's population living below the poverty live

The goal here is to create a shiny app to visualize this data. We want to be able to investigate the relationship between some of the variables on the dataset in a way that any person, regardless of their computer knowledge, can visualize the data, understand it, and create hypotheses about these countries and regions.

3.1 Global Definitions

We chose to build this app with three files. We use the mandatory `ui.R` and `server.R` files to define our app user interface and functions, respectively, but we also use `global.R` to have a more organized code.

In our app, the file `global.R` is used to load the packages needed and read the data that is analyzed. We need two packages for the app. The first package we load is the Shiny package. Without it, we could not run the app. The second package we load is Tidyverse. This is a collection of packages (notably `ggplot2` and `dplyr`, which are part of it) used to select, filter, and plot variables. Lastly, we import the data that we analyze. Since it is a .csv file, use the function `read_csv`, from package `readr`, which is loaded along with Tidyverse.

Putting this all together, these are the contents for the `global.R` file:

```
# contents of global.R file

library(shiny)
library(tidyverse)

wb <- read_csv(file = "data/world_bank.csv")
```

3.2 Interface Creation

After the packages and the data are loaded, the app interface has to be created. It needs to be easy to use. The more intuitive, the better. In our app, we want to choose variables from the World Bank dataset and plot their relationships. Therefore, dividing the app area into two parts is a simple but effective approach.

The first area is located to the left of the screen. The app must have menus for the user to select the variables that will be plotted. Moreover, it can have extra options to make the app more useful for its audience, which will be addressed later.

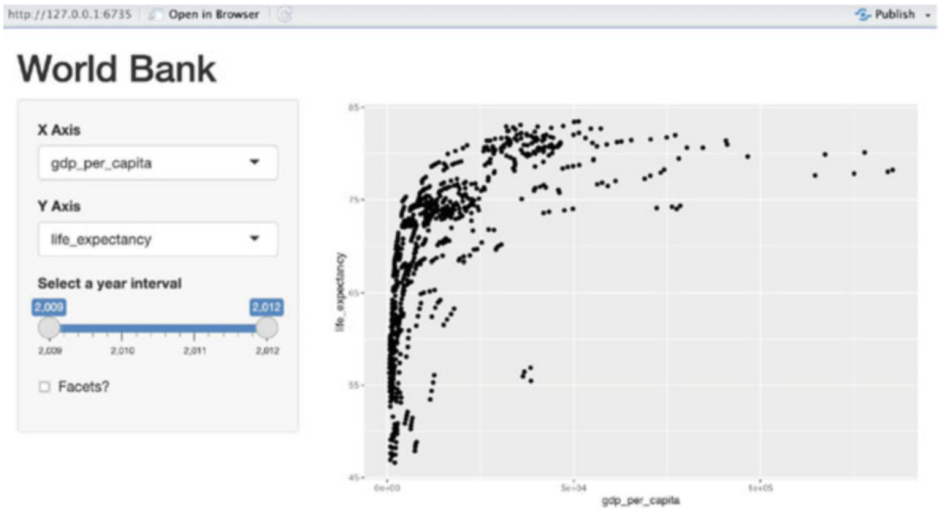
The second area is located at the right of the screen. This area will be used to plot the relationship between the variables. Since all the variables considered are quantitative, scatterplots are an excellent option to explore the dataset. Figure shows how the app is supposed to look.

The app interface has left and proper menus and plot areas, respectively.

It is interesting to note that the interface was created with only 17 lines of code. The complete code is shown below.

The majority of the functions are not familiar to the average R programmer. Here's a breakdown of what this code does:

- `pageWithSidebar`: The function that creates a page layout with a sidebar. It is used to divide the app into two parts. The sidebar is to the left, where the user selects its options, and the created plot is to the right.




```
pageWithSidebar(
  headerPanel('World Bank'),
  sidebarPanel(
    selectInput('xcol', 'X Axis', c("gdp_per_capita", "gdp"),
                selected = "gdp_per_capita"),
    selectInput('ycol', 'Y Axis', c("life_expectancy", "population"),
                selected = "life_expectancy"),
    sliderInput(inputId = "slider_year", label = "Select a year interval",
                min = 2009, max = 2012, step = 1, value = c(2009, 2012)),
    checkboxInput(inputId = "facet",
                  label = "Facets?",
                  value = TRUE)
  ),
  mainPanel(
    plotOutput('plot1')
  )
)
```

- `headerPanel`: A header panel is created with the title "World Bank", the app's name. This panel typically appears at the top of the page.
- `sidebarPanel`: A sidebar panel is created. In our example, this is where the users can input parameters for the visualization.
- `selectInput`: Define a dropdown menu allowing the selection of one or multiple items from a list of values. In this example, the menu it creates allows the users to select variables displayed on the x-axis (xcol) and y-axis (ycol). The options for the x-axis include "gdp_per_capita" and "gdp", with "gdp_per_capita" selected by default. The possibilities for the y-axis include "life_expectancy" and "population", with "life_expectancy" selected by default. Therefore, when the users access the app, they will see a scatterplot of gdp_per_capita versus life_expectancy.
- `sliderInput`: This function creates a slider input for selecting a range of years ("slider_year"). The range is from 2009 to 2012, the available years in the dataset, with a step size 1. The default range is from 2009 to 2012.

- `checkboxInput`: The checkbox input called facet is created here. It lets the user toggle whether to plot the graphs by year. By default, it is checked.
- mainPanel: This defines the main panel where the visualization output will be displayed.

Therefore, the code creates a user interface with widgets such as dropdown menus, a slider, and a checkbox for selecting variables, intervals, and options for visualization. The users can pick their preferred options, and these options will be used to generate a plot (namely `plot1`) in the main panel.

3.3 Server Definition

This section shows and explains the R code used to define the server part of our Shiny app. When used along the user interface defined in the previous section, it creates the plot based on user input. The code will be broken down in the following pages.

The function takes three parameters: input, output, and session, which are standard parameters for Shiny app functions. The `input` comes from the ui.R file, namely, from the input objects inside the `sidebarPanel`. The `output` is the object that will store the results of the server calculations, such as plots, tables, or pretty much any R object. The parameter `session` is optional and not used in this app. However, it can resume a Shiny app when the session ends abruptly if reconnections are enabled.

The first step in the server is to execute a data selection. In particular, a *reactive* data selection. This is obtained through the function `reactive`:

```
selected_data <- reactive({
  wb |>
    select(x = input$xcol, y = input$ycol, year) |>
    filter(year >= input$slider_year[1]) |>
    filter(year <= input$slider_year[2])
})
```

Notice that the contents of `reactive` function are only one use of `select` and two uses of `filter`, both performed by `dplyr` package. All the parameters for these functions come from the user interface. Hence, both columns selected come from the user choices from the dropdown menus, and the year limits are set by the slider.

The plot is created in the following next code excerpt.

The `renderPlot` function creates and renders a plot based on the filtered data. A conditional statement is tested if the `checkboxInput` is selected. If it is true, the function `facet_wrap` is added to the plot. If it is false, there are no facets in the final plot.

The plot is created using `ggplot2`. It is a scatter plot built on the object `selected_data`, created from the reactive function `selected_data`. Its contents,

```
output$plot1 <- renderPlot({
    if(input$facet){
      ggplot(selected_data(), aes(x = x, y = y)) +
        geom_point() +
        labs(x = as.character(input$xcol), y = as.character(input$ycol)) +
        facet_wrap(~ year)
    } else {
      ggplot(selected_data(), aes(x = x, y = y)) +
        geom_point() +
        labs(x = as.character(input$xcol), y = as.character(input$ycol))
      }
  })
```

as seen before, are based on the x and y variables selected by the user. Both plot labels are defined from the data frame column names.

Overall, the code builds a Shiny app that allows users to select two between four variables, specify lower and upper limits for the years a, and choose if the final plot will be split by year. The plot updates dynamically based on the user's selections.

Putting both the data filtering and the plot-making together, this is the server code obtained:

```
function(input, output, session) {

  # combine the selected variables into a new data frame

  selected_data <- reactive({
    wb |>
      select(x = input$xcol, y = input$ycol, year) |>
      filter(year >= input$slider_year[1]) |>
      filter(year <= input$slider_year[2])
  })

  # create a plot with a conditional plot

  output$plot1 <- renderPlot({
    if(input$facet){
      ggplot(selected_data(), aes(x = x, y = y)) +
        geom_point() +
        labs(x = as.character(input$xcol), y = as.character(input$ycol)) +
        facet_wrap(~ year)
    } else {
      ggplot(selected_data(), aes(x = x, y = y)) +
        geom_point() +
        labs(x = as.character(input$xcol), y = as.character(input$ycol))
      }
  })

}
```

With the files `global.R`, `ui.R`, and `server.R` finished, it is time to host the app.

4 Hosting

After the app is finished, it can be published on the Internet. Shiny apps can be run locally on a computer or in an intranet. However, they can live up to their potential if shared on the Internet. It is possible to self-host the app or use a free service to host it.

4.1 Self-hosting

When self-hosting is involved, the app is hosted on a cloud service paid for or maintained by someone. Advanced technical knowledge about creating and maintaining web servers and R is also needed. It gives the programmer more flexibility but comes with a cost. Usually, if the programmer needs to gain technical knowledge on these subjects, it is advised to use a free hosting service.

Federal University of Rio Grande do Norte, located in Brazil, has a Shiny server available for academic purposes. Any student or professor worldwide can submit their app, and if it is approved, the author will receive free hosting for their project. The project needs to be legal and has its code shared on a public repository at GitHub. The app built in this chapter can be found at http://shiny.estatistica.ccet.ufrn.br/world_bank_shiny/.

4.2 Free Hosting

Free hosting options are also available. When writing this chapter, shinyapps.io was one of the best options for hosting Shiny apps on the cloud. It offers easy deployment, scalability, security, and performance to anyone interested in sharing their Shiny app.

However, its free tier comes with some limitations. Those accounts unwilling to pay a monthly fee can host only 5 Shiny apps and have 25 active hours. Active hours are hours when the applications are not idle. That is, if 25 users run the apps for 1 hour each, the applications will be available again at the end of the month.

Different plans can be selected according to the available budget. Each option gives more slots for apps and more active hours. There are no plans with unlimited active hours, but the highest tier offers 10,000 active hours to be shared between an unlimited number of apps. This is usually more than enough for most academic and industrial Shiny apps.

Therefore, shinyapps.io is a very convenient way to host Shiny apps. It makes it easier for those who do not want or cannot have their server host their Shiny apps.

5 Conclusion

This chapter has shown how the Shiny package offers a powerful framework within the R programming language, enabling users to effortlessly create interactive web applications without the need for extensive knowledge of web development languages such as HTML, CSS, or JavaScript. By leveraging Shiny's key features like reactive programming, user interface components, server-side processing, web design templates, and software deployment capabilities, R users can easily build personalized reports, interactive visualizations, simulations, and data analysis interfaces.

The practical guide provided in this chapter illustrates the process of creating a Shiny app from scratch, demonstrating how to structure both the user interface and the server component. Through a step-by-step example of visualizing economic data from the World Bank, readers gain insight into the seamless integration of Shiny with R's statistical capabilities.

Moreover, the discussion on hosting options emphasizes the accessibility of Shiny apps to a global audience, whether through self-hosting on cloud services or utilizing free hosting platforms. This accessibility ensures that Shiny apps can be shared and utilized by a wide range of users, from statisticians and data scientists to educators and students, thereby enhancing data visualization, communication, and education in various domains.

In summary, the Shiny R package is a valuable tool for empowering users to create interactive web applications that unlock R's potential for data analysis, visualization, and communication, all within a user-friendly and intuitive framework.

References

World Bank. "Economic Data." *World Development Indicators*. The World Bank Group, 2024, https://data.worldbank.org/indicator/NY.GDP.MKTP.CD. Accessed 15 Feb. 2024.

Chang, Winston, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges. 2023. *Shiny: Web Application Framework for r*. https://CRAN.R-project.org/package=shiny.

Ekiz, H. Atakan, Christopher J. Conley, W. Zac Stephens, and Ryan M. O'Connell. 2020. "CIPR: A Web-Based R/Shiny App and R Package to Annotate Cell Clusters in Single Cell RNA Sequencing Experiments." *BMC Bioinformatics* 21 (1): 191. https://doi.org/10.1186/s12859-020-3538-2.

Jak, Suzanne, Terrence D. Jorgensen, Mathilde G. E. Verdam, Frans J. Oort, and Louise Elffers. 2021. "Analytical Power Calculations for Structural Equation Modeling: A Tutorial and Shiny App." *Behavior Research Methods* 53 (4): 1385–1406. https://doi.org/10.3758/s13428-020-01479-0.

R Core Team. 2023. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Sievert, Carson. 2020. *Interactive Web-Based Data Visualization with R, Plotly, and Shiny.* CRC Press.

Wickham, Hadley. 2021. *Mastering Shiny*. 1st ed. O'Reilly Media, Inc.
Wojciechowski, J., A.M. Hopkins, and R.N. Upton. 2015. "Interactive Pharmacometric Applications Using R and the Shiny Package." *CPT: Pharmacometrics & Systems Pharmacology* 4 (3): 146–59. https://doi.org/10.1002/psp4.21.