Shinny App

Data Tyding and Reporting

Sara Dovalo del Río and Javier Muñoz Flores

22/03/2022

What is Shiny?

Shiny is an R package which allows to users develop code and design small interactive programs through a web page. Its main goal is to produce applications in which there is a continuous interaction between the user and the server, in the sense that the user can interact with the displayed content and modify it according to a set of predefined rules.

A Shiny App is mainly structured in two parts:

- **UI**: The user interface takes care of the appearance and inputs of the application. Users create inputs through widgets, which are coded in the UI part of the Shiny application (buttons, checkboxes, sliders and text inputs).
- **Server**: Server contains the code where the results are generated, such as graphs, maps, values and tables and instructions for *R* on how the result should change based on the input in the user interface.

Our dataset and the goal of our aplication

We have chosen a dataset that captures the leading causes of death among New York City residents by sex and race between 2007 and 2014 in order to see how race and sex are related to causes of death and their evolution over time.

- `Year`: The year of death.
- `Leading Cause`: The cause of death.
- `Sex`: The decedent's sex.
- `Race Ethnicity`: The decedent's ethnicity.
- `Deaths`: The number of people who died due to cause of death.
- `Death Rate`: The death rate within the sex and Race/ethnicity category
- `Age Adjusted Death Rate`: The age-adjusted death rate within the sex and Race/ethnicity category.

Mainly, what our Shiny App does is that, given the dataset just described previously, it is able to calculate the most frequent causes of mortality (number of deaths, corresponding rate and corresponding age adjusted death rate) and plot it using a bar chart and a table.

The output of the application will be different depending on the mortality study year, sex and race that the user enters in the user interface.

Building our Shiny App

Preprocessing

First of all, what we have done is to "clean" our dataset so that our analysis will be clearer and easier (thinking about the later graphical representations that our application will make).

We have mainly done three things:

- Convert to numerical variables the number of deaths, the death rate and the age-adjusted death rate (variables `Deaths`, `Death Rate` and `Age Adjusted Death Rate`) as it will be necessary later to be able to correctly perform our analysis.
- Remove those levels of the categorical variables `Race Ethnicity` and `Leading Cause` that are of no interest to us when studying the number of deaths according to sex and age, such as: *Other Race/ Ethnicity* and *Not Stated/Unknown* of the variable `Race Ethnicity` and "All Other Causes" in the variable `Leading Cause`.
- To finish, we have renamed the different levels of the `Leading Cause` variable as the names were too long and we will do this by performing a pipeline on our data and applying `mutate()` by using the functions `unique()` (which returns all the different levels of this variable) and `str_replace_all()` (which works well as long as there are no brackets in the text) and, where in the first line what we do is to remove all the brackets from the variable and then rename each of the categories. Finally, we have renamed the variable names in the dataset to manipulate the data more easily using the `colnames()` command to become the variables `year`, `leading_cause`, `sex`, `race_ethnicity`, `deaths`, `death_rate` and `age_adjusted_death_rate` respectively.

Also note that we have saved our "clean" data in a new text file called `final_data.csv` which we will use later in our Shiny App.

User Interface (UI)

What our user interface will mainly do is three things:

1. It will give a title to our application using the `titlePanel()` command.
2. It will create widgets so that the user can enter the year, gender and race/ethnicity for which we want to see the leading causes of mortality. The `inputId` argument determines how you will refer to this widget when writing your server code, and the `label` argument is where you set the text that will appear above the widget in the application itself. We will now describe each of the widgets in detail:

- The first of them, referring to the year of study is a drop-down selector, which we will create using the `selectInput()` function, which will display a list of the different years considered in our dataset and the user will be able to choose one of them.
- The next two widgets will be related to the sex and race that the user wants to enter and will be of the type `radioButtons()` identifying the two options of the `sex`variable of our data set: F and M with *Female* and *Male* respectively. On the other hand, the possible options of our third widget referring to race will be the corresponding categories of our `race_ethnicity` variable and we will introduce them through the `unique(race_ethnicity)` command. These 3 inputs to our application will be inside the `selectbarLayout()` and `sidebarPanel()` commands since our application will have a sidebar with a single panel as user interface.

3. Last but not least, what our user interface does is to tell our application where our outputs or results will go (in this case the bar chart and the table with the mortality results for the corresponding year, gender and age selected) once we have created them later on in our application's server. This will be done by means of the function `mainPanel()` by referring to each of the outputs: the bar chart is called first using the `plotOutput()` function, and in quotes we name it "mortalityPlot" and the table using the `dataTableOutput()` function of the `DT` package and name it "mortalityTable".

All this code is contained in a `fluidPage()` command (see Appendix).

Server

As we said before, the server will contain the code where the results are generated and also where we specify how the application should react to the different inputs that users can make through the UI.

The server is a function that takes one input and one output argument, and all our code to create our server comes inside this function. We will mainly perform two steps to code our server:

1. First, inside the `reactive()` function (which filters our `final_data.csv` data based on the user's choice of widgets in the UI), we create an object called `user` and that three entries (as many as we have created in the UI). These names must exactly match the `inputId` names given in the UI.
2. Next we can create our two outputs: the bar chart and the table. Remember that the name we give to each output has to match exactly the name we gave it in the `mainPanel()` function of the UI.

- First we tell the application to make a plot with the `renderPlot()` function, and then we create this plot using `ggplot()` (where we use the data selected by the user named previously collected in the `user` object) where through the `reorder()` argument we assign the causes of death (variable `leading_cause`) ordered by decreasing number (indicated with the `-`) of deaths.
- Secondly, we use the `DT` package to create the table using the `renderDataTable()` function and the `datatatable()` function where we again use the `user` variable instead of our full dataset. Within these functions of the `DT` package we specify the four columns we are interested in: `leading_cause`, `deaths`, `death_rate` and `age_adjusted_death_rate`. As we did in the preprocessing part we use `colnames()` to rename the column names of our table to make it look cleaner. In the `options` argument, we indicate that the data should be sorted based on column ('deaths') in descending order as it will be more visual than the default alphabetical order.

Appendix

Preprocessing

```
if (!require("shiny")) install.packages("shiniy")
if (!require("dplyr")) install.packages("dplyr")
if (!require("ggplot2")) install.packages("ggplot2")
if (!require("readr")) install.packages("readr")
if (!require("tidyverse")) install.packages("tidyverse")
library(shiny)
library(dplyr)
library(ggplot2)
library(readr)
library(tidyverse)

# Load data
data = read_csv("death_NY.csv")
attach(data)

#-----------------------------------
# Preprocessing
#-----------------------------------
# We convert the variables of interest in our analysis into numerical variables
cbind(lapply(data,class))
data$Deaths <-  as.numeric(data$Deaths)
data$"Death Rate" <-  as.numeric(data$"Death Rate")
data$"Age Adjusted Death Rate" <- as.numeric(data$"Age Adjusted Death Rate")

# As the objective of this Shiny App is to obtain the causes of death of
#the NY population according to race and sex,
# we omit those values of the variable Race Ethnicity
#and Leading Cause that are not identified.
ind1 <- which(data$"Race Ethnicity"== "Other Race/ Ethnicity")
ind2 <- which(data$"Race Ethnicity"== "Not Stated/Unknown")
ind3 <- which(data$"Leading Cause"== "All Other Causes")
data <- data[c(-ind1,-ind2,-ind3),]

# To make the names shorter so that the graphical
#representations come out better
colnames(data) <- c("year", "leading_cause", "sex",
                    "race_ethnicity", "deaths", "death_rate",
                    "age_adjusted_death_rate")

unique(data$leading_cause)
data =
  data %>%
  mutate(
    leading_cause = str_replace_all(leading_cause, "[//(//)]", ""),
    leading_cause = str_replace(leading_cause, "Influenza Flu and Pneumonia J09-J18",
                                "Influenza & Pneumonia"),
    leading_cause = str_replace(leading_cause, "Accidents Except Drug Posioning V01-X39,
                                X43, X45-X59, Y85-Y86", "Accidents"),
```

```
    leading_cause = str_replace(leading_cause, "Cerebrovascular Disease Stroke: I60-I69",
                                "Cerebrovascular Disease"),
    leading_cause = str_replace(leading_cause, "Assault Homicide: Y87.1, X85-Y09", "Assault"),
    leading_cause = str_replace(leading_cause, "Essential Hypertension and Renal Diseases (I10, I12)",
                                "Hypertension & Renal Dis."),
    leading_cause = str_replace(leading_cause, "Human Immunodeficiency Virus Disease HIV: B20-B24",
                                "HIV"),
    leading_cause = str_replace(leading_cause, "Diseases of Heart I00-I09, I11, I13, I20-I51",
                                "Diseases of Heart"),
    leading_cause = str_replace(leading_cause, "Alzheimer's Disease G30", "Alzheimer's Disease"),
    leading_cause = str_replace(leading_cause, "Chronic Liver Disease and Cirrhosis K70, K73",
                                "Chronic Liver Disease/Cirrhosis"),
    leading_cause = str_replace(leading_cause, "Malignant Neoplasms Cancer: C00-C97",
                                "Malignant Neoplasms"),
    leading_cause = str_replace(leading_cause, "Diabetes Mellitus E10-E14", "Diabetes Mellitus"),
    leading_cause = str_replace(leading_cause, "Mental and Behavioral Disorders due to Accidental Poison
                                and Other Psychoactive Substance Use F11-F16, F18-F19, X40-X42, X44",
                                "Accidental Poisoning/Substance Use"),
    leading_cause = str_replace(leading_cause, "Septicemia A40-A41", "Septicemia"),
    leading_cause = str_replace(leading_cause, "Chronic Lower Respiratory Diseases J40-J47",
                                "Chronic Lower Respiratory Dis."),
    leading_cause = str_replace(leading_cause, "Nephritis, Nephrotic Syndrome and Nephrisis
                                N00-N07, N17-N19, N25-N27", "Nephritis"),
    leading_cause = str_replace(leading_cause, "Certain Conditions originating in the Perinatal Period
                                "Perinatal Period Conditions"),
    leading_cause = str_replace(leading_cause, "Viral Hepatitis B15-B19", "Viral Hepatitis"),
    leading_cause = str_replace(leading_cause, "Intentional Self-Harm Suicide: X60-X84, Y87.0",
                                "Suicide"),
    leading_cause = str_replace(leading_cause, "Congenital Malformations, Deformations,
                                and Chromosomal Abnormalities Q00-Q99", "Congenital Malformations")
  )

# We create a new clean dataset
View(data)
write_csv(data, "final_data.csv")
```

Shiny App

```
#-----------------------------------
# Our Shiny App
#-----------------------------------

data2 <- read_csv("final_data.csv")

ui <- fluidPage(

  # Application title
  titlePanel("Mortality in New York according
             to sex and race between years 2007-2014"),

  # Sidebar with a 3 inputs
```

```
  sidebarLayout(
    sidebarPanel(
      # selectInput(): , our choices are simply a list of all of
      #the years represented in our dataset
      selectInput(inputId = "year",
                  label = "Select Year:",
                  choices = c("2007",
                              "2008",
                              "2009",
                              "2010",
                              "2011",
                              "2012",
                              "2013",
                              "2014")),
      radioButtons(inputId = "sex",
                   label = "Sex:",
                   choices = c("Female" = "F", "Male" = "M")),
      radioButtons(inputId = "race",
                   label = "Race/Ethnicity:",
                   choices = unique(data2$race_ethnicity))
    ),
    # Show plot and table
    # mainPanel(): we're telling Shiny where they'll go
    # once we've created them
    mainPanel(
      plotOutput("mortalityPlot"),
      DT::dataTableOutput("mortalityTable")
    )
  )
)


server = function(input, output) {
  # reactive(): we first require the three inputs
  #that we created in the UI. These names should
  #align exactly with the inputId names that were created in the UI
  user = reactive({
    req(input$year)
    req(input$sex)
    req(input$race)
    filter(data2, year == input$year) %>%
      filter(sex %in% input$sex) %>%
      filter(race_ethnicity %in% input$race)
  })
  output$mortalityPlot = renderPlot({
    # reorder(): it makes more sense to have the
    #leading causes of death arranged by decreasing numbers of deaths
    # leading_cause variable should be ordered by decreasing
    #(specified with the "-") numbers of deaths
    # "stat = 'identity': we will provide the y-values directly
    #through our death variable
    ggplot(data = user(), aes(x = reorder(leading_cause, -deaths), y = deaths)) +
      geom_bar(stat = 'identity', color = 'lightpink1', fill = 'lightpink1') +
```

```
      labs(
        title = "Leading Causes of Death",
        x = "Causes",
        y = "Number of Deaths"
      ) +
      theme(axis.text.x = element_text(angle = 45, hjust=1))
  })

  output$mortalityTable =
    DT::renderDataTable({
      # "options": ordered based on deaths in descending order
      DT::datatable(user()[,c("leading_cause", "deaths",
                              "death_rate", "age_adjusted_death_rate")],
                    colnames = c("Leading Cause of Death",
                                 "Number of Deaths", "Death Rate", "Age-Adjusted Death Rate"),
                    options = list(order = list(2, 'des')),
                    rownames = FALSE,
      )

    })
}

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